80SM-000020-1

パーソナルコンピュータ

FMシリーズ

ユーザーズマニュアル
CP/M-86編

富士通

CP
ユー

まえがき

ＣＰ／M-86$^{T.M.}$は，8086および8088　16ビット・マイクロプロセッサ用に Digital　Research 社が設計したオペレーティンググ・システムである。

ＣＰ／M-86のファイル構造は，8080またはＺ80®マイクロプロセッサ・チップを基にしたコンピュータのためにDigital Research 社が開発したＣＰ／M®オペレーティング・システムのファイル構造と互換性がある．つまりディスクの形式が標準的な単密度形式と同じであれば，ＣＰ／M-86はＣＰ／Mと同じデータ・ファイルを読取ることができる。システム・コールはできるだけＣＰ／Mに近いものとし，すでに親しみのあるアセンブリ言語プログラミング環境を提供している．このためアプリケーション・プログラムは，ＣＰ／M-86のもとで実行できるように容易に変換することができる。

ＣＰ／M-86にとって必要なハードウェアの最小構成は，8086または8088マイクロプロセッサをもとにしたコンピュータ・システム，32ｋ（キロバイト）のランダム・アクセス・メモリ，キーボードおよび表示装置，それに一般的には２台の８インチ・フロッピー・ディスク・ドライブである。

一緒に提供されるユーティリティ・プログラムを除き，ＣＰ／M-86オペレーティング・システム自体が使用するメモリの大きさは，約12キロバイトである？ＤＤＴ-86$^{T.M.}$を実行するには，48ｋのメモリが必要であり，ＡＳＭ-86$^{T.M.}$やＣＰ／M-86のもとで実行されるアプリケーション・プログラムの多くを実行するには，64ｋのメモリが必要である．

これらの最小構成より大きなシステムを構築する場合には，ＣＰ／M86はそれぞれのコンピュータに追加できるその他の多くの機能をサポートする．例えば，ＣＰ／M-86は最大１メガバイトのランダム・アクセス・メモリ（ＲＡＭ）をサポートすることができるが，これは8086または8088マイクロプロセッサで可能な最大限のメモリである。またＣＰ／M-86は最大８メガバイトの記憶領域を持つ論理ディスク・ドライブを最大16台までサポートすることができ，その結果最大128メガバイトのオンライン記憶装置を使用できるようになる。

このマニュアルではＣＰ／M-86を紹介し，その使い方を示す。本書ではＣＰ／M-86システムが既に使われているものと仮定している。(ハードウェアとソフトウェアの間のインタフェースはＣＰ／M-86システム・ガイドに示されている指示に従って基本Ｉ／Ｏシステム（ＢＩＯＳ）で構成しなければならない)．本書ではまた読者がコンピュータの各部分に精通しており，そのセット・アップの仕方，電源の入れ方，またディスクの取扱い方について十分熟知しているものと仮定している．しかしコンピュータに関して豊富な経験を持っているとは仮定していない。

第１章では，ＣＰ／M-86の起動方法，コマンドの入力方法，それにバックアップ・ディスクの作成の件方について説明する．第２章では，ディスクとファイルについて説明する．

第３章では，第４章のコマンドのまとめを理解するのに必要なＣＰ／M-86のコマンドの概念を展開する．コマンドのまとめでは，ＣＰ／M-86と一緒に提供されるすべてのユーザ・プログラムについて説明する．

第５章では，ＣＰ／M-86のファイル・エディタ（ファイル編集プログラム）であるＥＤの使い方について説明する．ＥＤを使用すればプログラムやテキスト，それにデータ・ファイルを作成したり編集したりすることができる．

付録Ａには，ＡＳＣＩＩから16進数への変換テーブルが示されている．付録Ｂは，ＣＰ／M-86に関連するファイル型を示したものである．付録Ｃには，ＣＰ／M-86の制御文字が示されている．付録Ｄには，特殊な状況が発生した時にCP／M-86が表示するメーセージが示されている．それぞれの状況で修正が必要な場合には，どのような処置を取るべきかについても付録Ｄに示されている．付録Ｅは，一般的に使われるコンピュータ用語を集めた簡単な用語集である．

複雑なプログラムについては，ＣＰ／M-86プログラマ・ガイドとシステム・ガイドに示されている。ＡＳＭ-86はＣＰ／M-86のアセンブラである．アセンブリ言語プログラムを書くことを決定し，それぞれのコンピュータの8088マイクロプロセッサ命令セットを十分に理解するようになるまで，ＡＳＭ-86は不要である．アセンブリ言語でプログラムを書く場合には，ＡＳＭ-86を使用することにより8086または8088マイクロプロセッセ・プログラムの作成が簡略化されることがわかるだろう．ＤＤＴ-86は，ＣＰ／M-86のデバッギング・プログラムである．ＤＤＴ-86を使用すれば，高級言語やＡＳＭ-86で書かれたプログラムのエラーを検出することができる．

目　次

1. はじめに……1
1.1 CP／M-86の起動方法……1
1.2 コマンド行……2
1.3 CP／M-86行編集制御文字……4
1.4 ファイル・バックアップの必要性……5
1.5 CP／M-86ディスクのコピー方法……6

2. ファイル，ディスク，ドライブおよび装置……7
2.1 ファイルとは何か……7
2.2 ファイルの作成方法……7
2.3 ファイル名のつけ方……7
2.4 ファイルのアクセス……9
2.5 複数ファイルのアクセス……10
2.6 ファイル編成と保護……11
2.7 ディスクにファイルを記憶する方法……12
2.8 ディスクの交換……12
2.9 省略時ドライブの変更……13
2.10 その他のCP／M-86ドライブ機能……13
2.11 その他のCP／M-86の装置……14

3. CP／M-86のコマンドの概念……15
3.1 2種類のコマンド……15
3.2 組込みコマンド……15
3.3 トランジェント・ユーティリティ・コマンド……16
3.4 CP／M-86のコマンドの探索方法……17
3.5 制御文字コマンド……18

4. コマンドのまとめ……19
4.1 コマンドの各部分の名称の統一……19
4.2 コマンドの説明方法……21
4.3 ASM-86（アセンブラ）コマンド……24
4.4 COPYDISK（コピー・ディスク）コマンド……26
4.5 DDT-86（ダイナミック・デバッギング・ツール）コマンド……28
4.6 DIR（ディレクトリ）組込みコマンド……30
4.7 ED（テキスト・エディタ）コマンド……32
4.8 ERA（削除）組込みコマンド……37
4.9 GENCMD（CMDファイルの生成）コマンド……38
4.10 HELP（ヘルプ）コマンド……40
4.11 PIP（周辺装置変換プログラム）コマンド……42
4.11.1 単一ファイルのコピー……42
4.11.2 複数ファイルのコピー……44
4.11.3 ファイルの組合せ……45
4.11.4 補助装置との間でのファイル・コピー……46
4.11.5 多重コマンド・モード……47
4.11.6 PIPのオプションの使い方……48
4.12 REN（名前変更）組込みコマンド……51
4.13 STAT（ステータス）コマンド……52
4.13.1 ドライブの読取り専用ステータスのセット……52
4.13.2 ディスク未使用領域の表示……53
4.13.3 ファイル――使用領域とアクセス・モードの表示……54
4.13.4 ファイル・アクセス・モード（属性）のセット……56
4.13.5 ディスク・ステータスの表示……57
4.13.6 アクティブ・ファイルを持つユーザ番号の表示……57
4.13.7 STATコマンドと装置名の表示……58
4.13.8 物理装置の表示と論理装置への割当て……58
4.14 SUBMIT（バッチ処理）コマンド……60
4.15 TOD（日時の表示とセット）コマンド……62
4.16 TYPE（ファイルの表示）組込みコマンド……64
4.17 USER（ユーザ番号の表示とセット）組込みコマンド……65

5. ED CP／M-86エディタ……66
5.1 EDの紹介……66
5.2 EDの起動……67
5.3 EDオペレーション……68
5.3.1 テキストのバッファへの追加……69
5.3.2 EDからの退去……69
5.4 基本的な編集コマンド……71
5.4.1 文字ポインタの移動……72
5.4.2 メモリ・バッファの内容の表示……73
5.4.3 文字の削除……74
5.4.4 メモリ・バッファへの文字の挿入……75
5.4.5 文字の置換……76
5.5 EDコマンドの組合わせ……77
5.5.1 文字ポインタの移動……77
5.5.2 テキストの表示……77
5.5.3 編集……78
5.6 拡張EDコマンド……79
5.6.1 CPの移動とテキストの表示……79
5.6.2 文字ストリングの探索と置換……80
5.6.3 テキスト・ブロックの移動……82
5.6.4 変更データの保管と放棄：EDからの退去……83
5.7 EDエラー・メッセージ……85

付　録
A．ASCIIと16進数の変換テーブル……87
B．CP／M-86のファイル型……91
C．CP／M-86の制御文字……92
D．CP／M-86のエラー・メッセージ……93
E．用語集……101

第1章
はじめに

この章ではコンピュータおよびCP／M-86の基礎的なことについて説明する．まず，CP／M-86のスタートアップ手順と初期メッセージについて説明し，次にCP／M-86のコマンドの入力方法と，CP／M-86のオリジナル・ディスクのバックアップ用コピーの作り方について説明する．

CP／M-86はディスクに磁気的に記憶されている情報を，プログラムやデータから構成されるファイルにグループ分けすることにより管理する．CP／M-86はファイルを磁気ディスクからコンピュータ・メモリにコピーすることができるし，プリンタなどの周辺装置に出力することもできる．CP／M-86は，ユーザがキーボードから入力したコマンドをもとに，いろいろなプログラムを実行することによって，このようなタスク（作業）やその他のタスクを遂行する．

プログラムがメモリに記憶されると，そのプログラムは，特定のタスクを行なうようにコンピュータに指示するステップの集まりを実行する．CP／M-86を使って自分のCP／M-86プログラムを作成することもできるし，豊富なCP／M-86のアプリケーション・プログラムの中から適切なものを選択し，ゲームや教育に利用したり，事務処理および科学技術関係の問題を解くこともできる．

1.1 CP／M-86の起動方法

CP／M-86を起動する，あるいはロードするということは，CP／M-86のオリジナル・システム・ディスクをコピーしたディスクをコンピュータ・メモリに読み取ることである．

電源を入れ，CP／M-86システム・ディスクをドライブAに挿入する．ドライブAは一般に，コンピュータ・ユニットの右側に組込まれているドライブである．ディスクを挿入したら，ドライブの扉を閉じる．それからRESETまたはRESTARTボタンを挿すと，自動的にCP／M-86がメモリにロードされる．

電源を入れ，CP／M-86を再び起動したい場合には，まずCP／M-86システム・ディスクがドライブAにセットされていることを確認し，それからRESETまたはRESTARTボタンを押す．この手順をシステム・リセットまたは〝システムのブーティング〟と呼ぶ．

システム・リセットを行なうと，CP／M-86がメモリにロードされる．CP／M-86がメモリにロードされると，まず，次のメッセージが画面に表示される．

```
CP/M-86          Version V.V
Copyright (c) 1981 Digital Research Inc.
```

上記のV，Vで表されているバージョン番号は，現在使用中のCP／M-86のバージョンの主改訂レベルと副改訂レベルを示すものである．この後に，次の2文字のメッセージが表示される．

```
A>
```

A＞という記号は，CP／M-86の〝システム・プロンプト(入力指示)〟である．システム・プロンプトは，CP／M-86がキーボードから入力されたコマンドを読取れる状態であることを示すメッセージである．また，このシステム・プロンプトは，ドライブAが〝ディフォルト(省略時)〟ドライブであることも示す．つまり，CP／M-86に対して特に指示しない限り，CP／M-86はプログラム・ファイルやデータ・ファイルがドライブAにセットされているディスクに存在するものと考えるのである．

1.2 コマンド行

CP／M-86は，ユーザがキーボードからタイプした特定のコマンドにしたがって，特定のタスクを実行する．CP／M-86のコマンド行はコマンド・キーワード，任意の（必ずしも指定する必要のない）コマンド・テイル，それにキャリッジ・リターン（復帰）から構成される．コマンド・キーワードは，マイクロプロセッサが実行するコマンド（プログラム）を識別するものである．コマンド・テイルには，ファイル名やオプション，パラメータなどの，コマンドに対する特別な情報を指定することができる．コマンド行を終了するには，RETURNキーを押さなければならない．

キーボードから文字をタイプすると，その文字が画面に表示され，カーソル（位置インディケータ）は右へ移動する．誤った文字をタイプしてしまった場合，ターミナルにバックスペース・キーがあればそのキーを押し，またバックスペース・キーがなければCTRL-Hという文字を入力して，カーソルを左へ移動し，エラーを修正することができる．

キーワードとコマンド・テイルを入力する場合，大文字と小文字をどのように組み合せてもかまわない．CP／M-86は，コマンド行に含まれるすべての文字を大文字として取扱う．

一般に，システム・プロンプトが表示されたら，そのすぐ後にコマンド行をタイプする．しかし，CP／M-86では，プロンプトとコマンド・キーワードの間にスペースが存在してもかまわない．

コマンド・キーワードは2種類のコマンドのいずれか一方を識別する．2種類のコマンドとは，組込みコマンドとトランジェント・ユーティリティ・コマンドである．

組込みコマンドはCP／M-86の一部分としてメモリに常駐しており，直ちに実行することができる．一方，トランジェント・ユーティリティ・コマンドはプログラム・ファイルとしてディスクに記憶されている．したがって，これらのコマンドを実行するには，メモリにロードしなければならない．トランジェント・ユーティリティ・プログラム・ファイルのファイル名の最後にはCMDが付いているので，ディスクのディレクトリを見れば，どのファイルがトランジェント・ユーティリティ・プログラム・ファイルなのかがわかる．

トランジェント・ユーティリティの場合、CP／M-86はコマンド・キーワードだけを検査する．ユーザがコマンド・テイルも指定している場合，多くのユーティリティは固有のコマンド・ティルを必要としているので，CP／M-86はそのコマンド・ティルを全く検査せずに，ユーティリティに渡す．

では，組込みコマンドを使って，CP／M-86がコマンド行をどのように読み取るかを示すことにしよう．DIRコマンドは，ディスク・ファイルの名前を画面に表示するようにCP／M-86に指示するコマンドである．システム・プロンプトが画面に表示されたらDIRというキーワードをタイプし，コマンド・ティルは省略し，RETURNキーを押す．

```
A>DIR
```

CP／M-86はこのコマンドに対する応答として，ドライブAにセットされているディスクに記憶されているすべてのファイルの名前を出力する．たとえば，ドライブAにCP／M-86システム・ディスクがセットされているとすれば，下記のファイル名が他のファイル名と一緒に画面に表示されるはずである．

```
COPYDISK CMD
PIP      CMD
STAT     CMD
```

CP／M-86はつづりの正しいコマンド・キーワードだけしか認識しない．したがって，誤った文字をタイプし，それを修正しないでRETURNキーを押してしまうと，CP／M-86は入力されたコマンド行と疑問符（？）を画面に表示する．たとえば，DIRコマンドを下記のようにDJRと誤ってタイプすると，CP／M-86は次のように応答し，

```
A>DJR
DJR?
```

コマンド・キーワードを認識できないことを示す．

ＤＩＲコマンドでは，コマンド・テイルとしてファイル名を入力することができる．したがって，ＤＩＲとファイル名を一緒にタイプすれば，特定のファイルがディスク上に存在するか否かを確認できる．たとえば，トランジェント・ユーティリティ・プログラムＣＯＰＹＤＩＳＫ．ＣＭＤがシステム・ディスクに存在するか否かを調べるには，次のようにタイプする．

```
A>DIR COPYDISK.CMD
```

このコマンドが入力されると，ＣＰ／M-86は指定されたファイルの名前を出力するか（そのようなファイルが存在する場合），あるいはＮＯ　ＦＩＬＥというメッセージを出力する（そのようなファイルが存在しない場合）．

コマンド・キーワードとコマンド・テイルを区切るために，ＤＩＲの後に少なくとも１つのスペースをタイプしなければならない．スペースを全く入力しないこと，ＣＰ／M-86は次のような応答を返す．

```
A>DIRCOPYDISK.CMD
DIRCOPYDISK.CMD?
```

1.3 CP／M-86行編集制御文字

すでに説明したようにバックスペース・キーを使用すればタイプの誤りを修正することができる．しかしCP／M-86は下記の制御文字コマンドをサポートしており，これらはさらに効率よい編集を行なうのに役立つ．これらの制御文字を使用すれば，コマンド行やプログラムへの入力行を編集することができる．制御文字をタイプするには，CONTROLキー(CTRLというラベルが付けられていることがある）を押したままで必要な英字キーを押す．それから両方のキーを開放する．

表1-1．制御文字コマンド

コマンド	意　味
CTRL-E	カーソルを次の行の初めに移動するが，前の行の入力は消去しない．
CTRL-H	カーソルを左へ1文字分だけ移動し，その文字を削除する．バックスペース・キーと同じ機能を持つ．
CTRL-I	カーソルを次のタブ・ストップの位置に移動する．タブ・ストップは自動的に8桁ごとにセットされている．このコマンドはTABキーと同じ機能を持つ．
CTRL-J	カーソルを現在の行の左へ移動し，コマンド行をCP／M-86に送る．RETURNキーと同じ機能を持つ．
CTRL-M	カソソルを現在の行の左に移動し，コマンド行をCP／M-86に送る。RETURNキーと同じ機能を持つ．
CTRL-R	現在のカーソルの位置に＃をタイプし，カーソルを次の行に移動し，今までに既にタイプしたコマンドの一部をタイプしなおす．
CTRL-U	いままでタイプしたコマンド行に含まれるすべての文字を捨て，現在のカーソルの位置に＃をタイプし，カーソルを次のコマンド行に移動する．
CTRL-X	いままでタイプしたコマンド行に含まれるすべての文字を捨て，カーソルを現在の行のはじめに戻す．

この表からわかるように，制御文字の中には同じ意味を持つものがある．例えばCTRL-JとCTRL-MはRETURNキーを押したのと同じ効果を持つ．これらの3通りのキーストロークは処理のためにコマンド行をCP／M-86に送る．またCTRL-Hはバックスペース・キーを押したのと同じ効果を持つ．

1.4 ファイル・バックアップの必要性

人間は間違いを犯すことがあり，コンピュータも同様である．人間またはコンピュータの犯したエラーによって，貴重なプログラムやデータ・ファイルが破壊されることもある．たとえば，あるコマンドのタイプを誤ると，作成したばかりのプログラムを誤って消去してしまうかもしれない．同様の事故が電子部品の故障から発生することもある．

データ処理の専門家達は，貴重なファイルのコピーを作成することによってプログラムやデータが破壊されるのを避ける．新しいプログラムを購入したら必ず作業用のコピーを作成し，もとのプログラムは保管しておく．作業用コピーからプログラムが誤って消去されても，もとのファイルから容易にそのプログラムを復元することができる．

データ処理の専門家はまた，新しいプログラムを開発中にそのプログラムやデータ・ファイルのコピーを頻繁に作成する．コピーを作成する頻度はそれぞれのプログラムによって異なるが，一般的には，情報を再入力するのにコピーを作成するよりも10～20倍以上の時間がかかるような時にコピーを作成する．

バックアップ・ファイルは２通りの方法で作成することができる．つまり一度に１つのファイルのコピーを作成する方法と，ディスク全体のコピーを作成する方法である．どちらを選択するかは，バックアップの必要なファイルがディスク上にいくつ存在するかによって異なる．１つのファイルをコピーするには１分もかからないが，ディスク全体をコピーするには２～３分かかる．

ＣＰ／M-86システム・ディスクに記録されている情報を変更するためのコマンドについてはまだ説明していない．その説明を行う前にオリジナル・ディスクから作業用コピーを作成してみよう．

1.5 CP／M-86ディスクのコピー方法

CP／M-86ディスクをバックアップするには，バックアップ用の8インチ・プロッピー・ディスク，COPYDISKトランジェント・ユーティリティ・プログラム，それにCP／M-86ディスクを使用する．

バックアップ・ディスクは新しいものでもすでに使用したものでもかまわない．8インチ・ディスクの中には右下のノッチが切り取られたものがある．このノッチはデータがディスクに書込まれることを防止するものであり〝書込み保護〟ノッチと呼ばれる．このようなディスクにデータをコピーするには書込み保護ノッチの上に小さなタブを置くことにより〝書込み可能〟にしなければならない．これらのタブはディスクと一緒に提供されている．

ディスクを使用する場合には，それぞれのコンピュータに付随するディスク・フォーマティング・プログラムを使って，新しいディスクのフォーマティングを行なったり，使用済ディスクを再フォーマティングする必要がある．ディスクが使用済のものなら，必要な情報がそのディスクに記憶されていないかどうかを確認する．COPYDISKはソース・ディスクから目的ディスクへブランクも含めてすべての情報をコピーするので，目的ディスクに記憶されていた情報はすべて失なわれてしまう．

CP／M-86ディスクのコピーを作成するには，COPYDISKユーティリティを使用する．まずシステム・ディスクをドライブAにセットし，フォーマティングされたディスクをドライブBにセットする．次にシステム・プロンプトが出力されたら，下記のコマンドを入力し，最後にキャリッジ・リターンを入力する．

```
A>COPYDISK
```

するとCP／M-86はCOPYDISKをメモリにロードしそれを実行する．
COPYDISKは画面に下記のメッセージを表示する．

```
CP/M-86 Full Disk COPY Utility
     Version 2.0

Enter Source Disk Drive (A-P) ?A

Destination Disk Drive (A-P) ?B

Copying Disk A: to Disk B:
Is this what you want to do (Y/N) ?Y

Copy started
Reading Track 0...
Copy completed.

Copy another disk (Y/N) ?N
Copy program exiting

A>
```

これでドライブBにあるディスクにCP／M-86のオリジナル・ディスクが正しくコピーされたので，ドライブAからオリジナル・ディスクを取り出し，それを安全な場所に保管する．オリジナル・ディスクが安全にそして変更されない状態で保管されていれば，作業用コピーに何にかが起っても，CP／M-86プログラム・ファイルは容易に復元することができる．

次にドライブBからコピー・ディスクを取り出し，それをドライブAに挿入する．そのディスクを自分のCP／M-86システム・ディスクとして使用し，さらに別のバックアップ・コピーを作成したり，本書に示されているいろいろな例を実際にやってみたり，またコンピュータの電源を入れ，CP／M-86の起動を行なってください．

第2章
ファイル，ディスク，ドライブおよび装置

CP／M-86の最も重要な仕事は，ディスクに記憶されているファイルをアクセスし維持することである．CP／M-86は，プログラム・ファイルやデータ・ファイルの作成，読取り，書込み，コピーそれに消去を行なうことができる．この章ではファイルとは何か，ファイルの作成方法，名前の付け方，アクセス方法，それにファイルをディスクにどのように記憶するかについて説明する．またディスクを変更したことやディフォルト・ドライブを変更したいことを，CP／M－86にどのように示せばよいかについても説明する．

2.1　ファイルとは何か

CP／M-86のファイルは，ディスクに格納されている関連情報の集まりである．どのファイルにも固有の名前が必要である．なぜなら，その名前を使ってファイルがアクセスされるからである．各ディスクにはディレクトリも記憶されておりディレクトリには，そのディスクに格納されているファイル名のリストと各ファイルの位置が示されている．

一般にファイルには，プログラム・ファイルとデータ・ファイルの2種類がある．プログラム・ファイルは実行可能なファイルであり，コンピュータが1ステップずつ実行することのできる一連の命令の集まりである，データ・ファイルは情報の集まりである．例えば名前と住所の表，店舗の在庫，ある会社の会計記録，書類テキストあるいは同様の関連した情報などがある．例えばコンピュータは，名前と住所を〝実行〟することはできないが，郵送用ラベルに名前と住所をプリントするプログラムを実行することはできる．

データ・ファイルには，プログラムのソース・コードも含まれている．一般にプログラムのソース・ファイルを実行可能なプログラム・ファイルにするには，前もってアセンブラまたはコンパイラで処理することが必要である．ほとんどの場合，プログラムを実行するとデータ・ファイルが処理される．しかしプログラムが実行可能なプログラム・ファイルを処理する場合もある．例えば実行可能なコピー・プログラムであるPIPは，1つまたは複数のコマンド・プログラム・ファイルをコピーすることができる．

2.2　ファイルの作成方法

ファイルを作成するには多くの方法がある．既存のファイルを新しい別の位置にコピーし，それに新しい名前を付けることによって，ファイルを作成することができる．CP／M-86のもとでは，トランジェント・ユーティリティPIPを使って，ファイルをコピーし，名前を付け変えることができる．ファイルを作成する2番目の方法は，テキスト・エディタを使用する方法である．

CP／M-86テキスト・エディタ，EDはファイルを作成し，ユーザの指定した名前をそのファイルに割当てることができる．最後にASM-86のように，一部のプログラムは入力ファイルを処理した結果として出力ファイルを作成する．

2.3　ファイル名の付け方

CP／M-86は，すべてのファイルをその固有なファイル記述によって識別する．ファイル記述（filespec）は3つの部分に分けることができる．

d：	ドライブ記述子	1文字，任意
ファイル名	ファイル名	1－8文字
typ	ファイル型	0－3文字，任意

ファイル記述は，英字と数字でおこなうことが望ましい．ＣＰ／M-86コマンド・プロセッサは下記の特殊文字を区切り文字として認識するので，これらの文字をファイル名やファイル型の中に指定してはならない．

< > . , ; : = ? * [ ]

ファイル記述は，下記のような１文字から８文字までのファイル名だけで構成することもできる．

MYFILE

ファイル名を作成する場合には，そのファイルに含まれる情報を示すような名前を付けるようにする．例えば顧客名のリストをファイルに格納する場合には，そのファイルには次のような名前を付けることができる．

CUSTOMER

名前は８文字以下の英数字で構成され，ファイルに何が記憶されているかをユーザに示すことができる．

ＣＰ／M-86のもとでコンピュータを使い始めると，ファイルが自然にファミリに分類されることがわかる．ファイル・ファミリを区分できるように，ＣＰ／M-86ではファイル型と呼ばれる１文字から３文字の任意のファミリ名をファイル名に追加することができる．ファイル型をファイル名に追加する場合には，ファイル名とファイル型の間にピリオドが必要である．３文字を使ってファイルのファミリを示すファイル型を作成してみよう．例えば顧客名のリストを格納しているファイルには，下記のようなファイル型を追加することができる．

CUSTOMER.NAM

ＤＩＲコマンドに対する応答としてＣＰ／M-86がファイル記述を表示する場合には，ファイル型を速かに比較できるように，短いファイル名やファイル型にはブランクが充塡される．

ＣＰ／M-86がディスクからメモリにロードする実行可能なプログラム・ファイルには，異なるファイル名が付けられている．これらはすべてＣＰ／M-86のもとで実行される8086または8088プログラムのファミリに分類される。ファイル型ＣＭＤは実行可能なプログラムのファミリを示すものである．

ＣＰ／M-86では，既にいくつかのファイル・ファミリが確立されている．下記の表は，ファイル型の一部とそれぞれのファミリの簡単な説明を示したものである．

表2-1．ＣＰ／M-86のファイル型

ファイル型	意　　味
ＣＭＤ	8086または8088機械語プログラム
ＢＡＳ	ＣＢＡＳＩＣソース・プログラム
＄＄＄	テンポラリ・ファイル
Ａ86	ＡＳＭ-86ソース・ファイル
Ｈ86	16進形式のアセンブルされたＡＳＭ-86プログラム
ＳＵＢ	ＳＵＢＭＩＴによって実行されるコマンドのリスト

2.4 ファイルのアクセス

コマンド・テイルでファイル記述をタイプする場合，組込みユーティリティまたはトランジェント・ユーティリティは，システム・プロンプトによって示されたドライブにセットされているディスクから指定されたファイルを捜す。例えば，次のコマンドをタイプすると，

```
A>dir copydisk.cmd
```

CP／M-86はドライブAにセットされているディスクのディレクトリを見てCOPYDISK．CMDを捜す．しかし別のドライブ，例えばドライブBにセットされているディスクをアクセスするように，CP／M-86に対して指示する方法が必要である．この理由から，CP／M-86ではドライブ記述子の付いた ファイル名を指定することができる． ドライブ記述子は，ドライブを示す英字とコロンから構成される．例えば，次のコマンドをタイプすると，

```
A>dir b:myfile.lib
```

CP／M-86は，ドライブBにセットされているディスクのディレクトリからファイルMYFILE．LIBを捜す．

またコマンド・キーワードとしてプログラム・ファイル名を使っている場合でも，ドライブ記述子の指定された実行可能なプログラム・ファイル名を入力することができる．例えば，次のコマンドをタイプすると，

```
A>b:pip
```

CP／M-86はドライブBにセットされているディスクのディレクトリから，ファイルPIP．CMDを捜す．CP／M-86がドライブBのディスクからPIPを見つけたら，そのPIPプログラムをメモリにロードし実行する．

ファイル名とファイル型はディスク・ディレクトリに記憶されているが，ドライブ記述子はディスクをあるドライブから別のドライブに移し変えることもあるので，変化する．したがって，ディスクをあるドライブから別のドライブに変更する場合には，1つのファイルに異なるファイル記述が割り当てられることになる．

2.5 複数ファイルのアクセス

特殊な〝ワイルド・カード（wild card）—曖昧なファイル参照を可能にする〞文字がファイル名またはファイル型に含まれている場合は，特定のＣＰ／Ｍ-86組込みユーティリティおよびトランジェント・ユーティリティは，複数のファイルを選択し処理することができる。ワイルド・カードを含むファイル記述は，ＣＰ／Ｍ-86に対し一致しなければならないパターンを与えるので，複数のファイルを参照することができる．ＣＰ／Ｍ-86はディスク・ディレクトリを探索し，ファイル名，またはファイル型がパターンと一致するファイルを選択する．

ワイルド・カード文字としては？と＊の２文字がある．？は同じ位置にあるどれか１つの文字と一致するものであり，＊はその位置にあるすべての文字と一致する．そしてその他の文字は，ファイル名あるいはファイル型に指定されているままである．ワイルド・カードを使用する場合の規則は下記の通りである．

- ？はスペースも含めて名前の中のどの文字とも一致する．
- ＊はファイル名またはファイル型の最後の文字，または唯一の文字でなければならない．ＣＰ／Ｍ-86はファイル名またはファイル型の終りにある＊を？と置き換える．
- 対応するファイルル名が８文字より短い場合には，ＣＰ／Ｍ-86はその名前をスペースで終るものとして取扱う．
- 対応するファイル型が３文字より短い場合には，ＣＰ／Ｍ-86はそのファイル型がスペースで終るものとして取扱う．

例えば，下記の６つのファイルを格納しているディスクがあるとしよう．

A.CMD, AA.CMD, AAA.CMD, B.CMD, A.A86, and B.A86

ワイルド・カードを含む名前が，これらのファイルのすべてあるいは一部と一致する例が下記に示されている．

.	????????.??? として取扱われる．
????????.???	６つの名前すべてと一致する．
*.CMD	????????.CMD として取扱われる．
????????.CMD	最初の４つの名前と一致する．
?.CMD	A.CMD および B.CMD と一致する．
?.*	?.??? として取扱われる．
?.???	A.CMD, B.CMD, A.A86, および B.A86 と一致する．
A?.CMD	A.CMD および AA.CMD と一致する．
A*.CMD	A???????.CMD として取扱われる．
A???????.CMD	A.CMD, AA.CMD, および AAA.CMD と一致する．

ＣＰ／Ｍ-86がワイルド・カード・パターンを使用するのはディスク・ディレクトリを探索する場合だけであり，したがってワイルド・カードはファイル名とファイル型でしか使用できない．ワイルド・カードをドライブ記述子で使用することはできない．

2.6 ファイル編成と保護

CP／M-86のもとではファイルを複数のグループに編成し，そのファイルが誤って変更されないように保護し，DIRコマンドに対する応答としてファイルがどのように表示されるかを指定することができる。CP／M-86はユーザ番号と属性をファイルに割当て，それをディスクのディレクトリに記録することによってこのような機能をサポートする．

ユーザ番号を使用すれば，ファイルを16ファイル・グループに分類することができる．ファイルはすべて0から150範囲のユーザ番号によって識別される．CP／M-86はファイルが作成されるときに，そのファイルにユーザ番号を割り当てる．コマンド・プログラムPIPを使ってファイルを別のユーザ番号にコピーしない限り，そのファイルには〝現在の〟ユーザ番号が割当てられる．組込みコマンドUSERを使用すれば，現在のユーザ番号を表示し変更することができる．

大部分のコマンドは，現在のユーザ番号を持つファイルだけしかアクセスすることができない．例えば，現在のユーザ番号が7の場合，DIRコマンドはユーザ番号7のもとで作成されたファイルだけを表示する．この例外はPIPコマンドである．〔Gn〕オプションが指定されている場合，PIPはあるユーザ番号を持つファイルをコピーし，そのコピーに別のユーザ番号を割当てることができる．

ファイル属性は，ファイルがどのようにアクセスできるかを制御するものである．ファイル・アクセス属性には2種類のものがある．

DIR／SYS属性は，DIR（ディレクトリ）またはSYS（システム）のいずれかに対してセットすることができる．ファイルを作成すると，そのファイルには自動的にDIR属性が付けられる．DIRコマンドは現在のユーザ領域の中に存在し，ユーザ番号が0，1，2，3または15であるファイルだけを表示する．

STATトランジェント・ユーティリティ・コマンドを使用すれば，ファイルにSYSまたはDIR属性を割当てることができる．DIRコマンドではSYS属性の付いたファイルを表示することができない．従ってSYSファイルを表示するには，DIRSコマンドを使用しなければならない．ただし，DIRSは現在のユーザが番号を持つシステム・ファイルだけを表示する．STATコマンドはSYS属性の割当てられたファイルを表示する．この場合も，STATは現在のユーザ番号を持つファイルだけを表示する．

SYS属性をユーザ番号0のファイルに割当てておけば，非常に役立つ．これらのファイルはコマンド・ファイルであり，ファイル・タイプがCMDのファイルである．コマンド・ファイルにユーザ番号0とSYS属性を割当てると，同じドライブのどのユーザ番号からもそのファイルを読取り，実行することができる．この機能を利用すれば，一般に使用されるプログラムをどのユーザ番号からでもアクセスできるようにすることができ，各ユーザ番号ごとに各コマンド・プログラムのコピーを作成する必要がなくなる．

RW／ROファイル・アクセス属性は，RW（読み書き）またはRO（読取り専用）のいずれかにセットすることができる．RW属性を持つファイルは，ディスクが書込み保護されていないか，あるいはディスクがセットされているドライブが読取り専用にセットされていないかぎり，いつでも読取りまたは書込みを行なうことができる．ファイルにRO属性が割当られている場合には，そのファイルにデータを書込もうとするとRead-Onlyというエラー・メッセージが表示される．したがって重要なファイルを保護したい場合，RO属性を使用することができる．

STATトランジェント・ユーティリティ・プログラムを使用すれば，ファイルまたはファイル・グループに対し，読み書き属性または読取り専用属性を割当てることができる．STATはまた，ドライブに読取り専用属性を割当てることもできる．この場合CTRL-Cを使用すれば，ログインされているドライブはすべて読み書き属性にリセットされる．

2.7　ディスクにファイルを記憶する方法

CP／M-86はディレクトリと呼ばれるディスクの特殊な領域に，各ファイルのファイル名，ファイル型，ユーザ番号それに属性を記録する．CP／M-86はまた，どのディスク・セクタがどのファイルに属しているかもディレクトリに記録する．ディレクトリは最大64のファイルに対して，このようなデータを格納できる大きさを持っている．

CP／M-86は，レコードがファイルに追加されるとき，そのファイルに対するディレクトリおよび記憶空間を割振る。ファイルを消去すると，CP／M-86は2通りの方法で記憶空間を再要求する．まずファイルのディレクトリ空間を開放し，別のファイルをカタログできるようにし，次にファイルの記憶空間を開放し，後で使用できるようにする．これは〝動的割合て（ダイナミック・アロケーション）〟と呼ばれ、CP／M-86の強力な機能である．CP／M-86は必要に応じて自動的に記憶空間をファイルに割当て，ファイルが消去される場合には，再割当てのためにその記憶空間を開放するので，ファイルがどれ位の大きさになるのかをCP／M-86に対して示す必要はない．

2.8　ディスクの交換

処理したいファイルを格納しているディスクがドライブに挿入されていなかったり，そのドライブが使用可能な状態でない場合には，CP／M-86はそのファイルに対して何も行なうことができない．ディスクがドライブにセットされている場合，そのディスクは〝オンライン〟であると呼ばれ，CP／M-86はそのディレクトリをアクセスすることができる．

場合によっては，ディスクをドライブから取り出し，別のファイルを格納している別のディスクを挿入しなければならないことがある．システム・プロンプトがコンソールに表示されている場合には，オンライン・ディスクを入換ることができる．しかしディスクに書込みを行ないたい場合には，システム・プロンプトのすぐ後にCTRL-Cをタイプすることにより，CP／M-86に対し，ディスクを入換たことを通知しなければならない．それに対する応答として，CP／M-86は新しいディスクに合わせてドライブをセットしなおす．

ディスクを入換た後，CTRL-Cをタイプし忘れると，CP／M-86は自動的に新しいディスクを保護する．したがってテキスト・エディタや コピー プログラムを使って新しいディスクに書込みを行なおうとすると，CP／M-86はオリジナル・ディスクがドライブに存在しないことを示し，次のメッセージを表示する．

```
Bdos err on d: RO
```

d：は新しいディスクのドライブ記述子である．このメッセージが表示されたら，CTRL-Cを1回タイプし，システム・プロンプトが表示されてから，さらにもう1回CTRL-Cをタイプして新しいディスクをログインしなければならない．

2.9 省略時ドライブの変更

ＣＰ／Ｍ-86を使用する場合には，必ず省略時ドライブと呼ばれるドライブが存在する．コマンド行にドライブ記述子を指定しない限り，ＣＰ／Ｍ-86およびユーティリティは省略時ドライブにセットされているディスクのディレクトリから，すべてのプログラムおよびデータ・ファイルを探索する．ＣＰ／Ｍ-86のシステム・プロンプトは省略時ドライブを示している．例えば次のメッセージは，

A>

ドライブAが省略時ドライブであることを示している．コマンドをＣＰ／Ｍ-86に入力する場合，どのディスクが省略時ドライブにセットされているかを覚えておかなければならない．さらにドライブ記述子を加えない場合には，アプリケーション・プログラムがどのファイルをアクセスするのかも知っておかなければならない．

ＣＰ／Ｍ-86を起動したときの省略時ドライブは，一般にドライブAである．複数のドライブがある場合には，省略時ドライブを変更することができる．省略時ドライブを変更するには，システム・プロンプトの後に希望する省略時ドライブのドライブ記述子をタイプし，ＲＥＴＵＲＮキーを押す．

A>B:

例えば，このコマンドは省略時ドライブをBに変更する．再び省略時ドライブを変更しない限り，システム・プロンプト・メッセージはすべて次のように表示される．

B>

このシステム・プロンプトは，ＣＰ／Ｍ-86およびそのユーティリティが，ファイル記述にドライブ記述子のないファイルに対しては，ドライブBにセットされているディスクのディレクトリを調べることを示している．

2.10 その他のＣＰ／Ｍ-86ドライブ機能

ＣＰ／Ｍ-86のもとでは，ファイルにＲＯ属性を与えることができるように，ドライブにもＲＯ属性を与えることができる．ドライブの省略状態はＲＷであるが，ドライブにセットされているディスクを入換ると，ＣＰ／Ｍ-86はそのドライブにＲＯ属性を割当てる．ドライブにＲＯ属性を割当てるには，第4章に示されているＳＴＡＴトランジェント・ユーティリティを使用する．ドライブをＲＷに戻すには，システム・プロンプトのすぐ後にＣＴＲＬ-Ｃをタイプしなければならない．

2.11 その他のＣＰ／M-86装置

ＣＰ／M-86は，コンピュータに接続されているすべての周辺装置を管理する．周辺装置としてはディスク・ドライブなどの外部記憶装置，キーボードやモデムなどの入力装置，それにプリンタやモデムおよびスクリーンなどの出力装置がある．

入出力装置の状態を追跡するために，ＣＰ／M-86は"論理"装置を使用する．下記の表はＣＰ／M-86の論理装置名を示したものであり，それぞれの装置が入力装置であるか出力装置であるかも示している．

表2-2．ＣＰ／M-86の論理装置

装置名	装置タイプ
ＣＯＮ：	コンソール．入力および出力
ＡＸＩ：	補助装置．入力
ＡＸＯ：	補助装置．出力
ＬＳＴ：	リスト装置．出力

ＣＰ／M-86は，物理的な装置を論理装置名と関連付ける．例えば，省略時コンソール入力装置はキーボードであり，省略時コンソール出力装置はスクリーンである．ＣＰ／M-86に任意の周辺装置を管理して欲しい場合には，ＳＴＡＴコマンドを使って代替周辺装置に論理装置名を割り当てなければならない．例えば，ＳＴＡＴコマンドはコンソール入力装置をキーボードからテレタイプに変更すること，またプリンタをＬＳＴ：という論理出力装置名に割当てることができる．

１つの論理入力装置は，１つの物理装置にだけしか割当てることができない．論理出力装置は，１つの物理装置にしか割当てることができない．さらに詳しい説明については，第４章のＳＴＡＴコマンドの説明を参照されたい．

第3章
CP／M-86のコマンドの概念

第1章で説明したように，CP／M-86のコマンド行はコマンド・キーワード，省略可能なコマンド・テイルそれにキャリッジ・リターンから構成される．この章ではコマンド・キーワードが識別できる2種類のプログラムについて説明し，CP／M-86がディスク上のコマンド・ファイルをどのように捜すのかについて，その方法を示す．またCP／M-86に対していろいろなタスクを実行するように指示する制御文字についても説明する．

3.1　2種類のコマンド

コマンド・キーワードは，CP／M-86の一部としてメモリに常駐するプログラムと，プログラム・ファイルとしてディスクに格納されているプログラムとを識別する．コマンド・キーワードがメモリに常駐するプログラムを識別する場合，そのプログラムは組込みコマンドと呼ばれる．コマンド・キーワードがディスクに格納されているプログラム・ファイルを識別する場合，そのプログラムはトランジェント・ユーティリティまたは単にユーティリティと呼ばれる．

CP／M-86には6つの組込みコマンドと16のトランジェント・ユーティリティが含まれている．ユーティリティをシステムに追加するには，CP／M-86と互換性のあるいろいろなアプリケーション・プログラムを購入すればよい．またプログラムとしての経験が豊富であれば，CP／M-86で動作する独自のユーティリティを作成することができる．

3.2　組込みコマンド

組込みコマンドはCP／M-86の一部であり，どのドライブにどのディスクがセットされているかということとは無関係に，いつでも使用することができる．組込みコマンドはCP／M-86の一部としてメモリに常駐しているので，ユーティリティよりも速やかに実行することができる．下記の表に示されている組込みコマンドの詳細については第4章で説明する．

表3-1．組込みコマンド

コマンド	意　　　味
DIR	ディスク・ディレクトリからDIR属性を持つファイル名のリストを表示する．
DIRS	SYS属性を持つファイルのファイル名のリストを表示する．
ERA	ディスク・ディレクトリからファイル名を削除し，そのファイルが占有していた記憶領域を開放する．
REN	ファイルの名前を付換える．
TYPE	スクリーンに文字ファイルの内容を書出す．
USER	ユーザ番号を他の番号に変更する．

3.3 トランジェント・ユーティリティ・コマンド

トランジェント・ユーティリティ・コマンドを実行するプログラムは，ユーザがそのコマンドを要求したときにだけメモリにロードされる．下記の表に示されている標準的なＣＰ／Ｍ-86のユーティリティの詳細については第５章で説明する．

コマンド	意　味
ＡＳＭ86	8086アセンブリ言語プログラムを機械コード形式に変換する．
ＣＯＰＹＤＩＳＫ	ＣＰ／Ｍ-86，プログラム・ファイルおよびデータファイルを含むディスクのコピーを作成する．
ＤＤＴ86	プログラムの検査を行ない，対話方式で“バグ”およびプログラミング・エラーを修正する．
ＥＤ	いろいろなプログラムでアクセスできるように文字ファイルを作成，変更する．
ＧＥＮＣＭＤ	ＡＳＭ-86の出力を用いて実行可能なコマンド・ファイルを作成する．
ＨＥＬＰ	ＣＰ／Ｍ-86の各コマンドの使い方に関する情報を表示する．
ＰＩＰ	ファイルを結合，及びコピーする．
ＳＴＡＴ	ファイルやディスクのステイタスを表示したり変更する．物理I/OデバイスをCP／Mの論理デバイスに割りつける．
ＳＵＢＭＩＴ	コマンドのファイルをCP／M-86に送りこみ，自動的にコマンドを実行させる．
ＴＯＤ	システムの日付および時刻をセットし表示する．

3.4 CP／M-86のコマンドの探索方法

コマンド・キーワードが組込みコマンドを識別しない場合，CP／M-86は省略時ドライブまたは指定されているドライブからプログラム・ファイルを見つける．ファイル名がキーワードと等しく，ファイル型がCMDであるファイルを見つける．例えば，次のコマンド行を入力したとしよう．

```
A>ED MYPROG.BAS
```

この場合CP／M-86は下記のステップでコマンドを実行する．

1） CP／M-86はキーワードEDが組込みコマンドでないことを検出する．

2） CP／M-86は，省略時ドライブのディレクトリからED．CMDというユーティリティ・プログラム・ファイルを見つける．現在のユーザ番号のもとでそのファイルを見つけることができない場合には，ユーザ番号0のもとでSYS属性をもつED．CMDというファイルを見つける．

3） CP／M-86がED．CMDを見つけると，そのプログラムをメモリにコピーし，制御をEDに渡す．

4） EDから抜け出すためのコマンドを入力するまで，EDは動作可能な状態である．

5） CP／M-86はシステム・プロンプトを出力し，ユーザが別のコマンド行を入力するのを待つ．

CP／M-86が組込みコマンドまたはトランジェント・ユーティリティを見つけることができない場合，ユーザのタイプしたコマンド行と疑問符（？）をスクリーンに表示することにより，キーワード・エラーを報告する．このようなエラー・メッセージが表示された場合，次の3通りのエラーのいずれかが発生したと考えられる．

・ キーワードが組込みコマンドでない．

・対応するCMDファイルが現在のユーザ番号のもとにも，ユーザ番号0のもとのSYS属性にも存在しない．

・ ドライブ記述子を指定した場合には，対応する CMDファイルが，指定されたドライブの現在のユーザ番号のもとにも，ユーザ番号0のSYS属性を持つものとしても存在しない．

例えば，省略時ディスクに標準的なCP／M-86のユーティリティだけしか含まれていない場合に，次のコマンド行をタイプしたとしよう．

```
A>EDIT MYPROG.BAS
```

この場合にはCP／M-86はエラーを報告するためのステップを実行する．

1） CP／M-86はまずキーワードEDITを調べ，それが組込みコマンドの1つでないことを検出する．

2） CP／M-86は省略時ディスクのディレクトリを調べ，まず現在のユーザ番号のもとでEDIT．CMDを捜し，見つからない場合にはユーザ番号0のもとでSYS属性を持つEDIT．CMDを見つける．

3） ファイルを見つけることができない場合，CP／M-86はスクリーンに次のメッセージを出力し，

EDIT？

そのコマンドが実行できないことを示す．

4） CP／M-86はシステム・プロンプトを表示し，ユーザが別のコマンド行を入力するのを待つ．

3.5 制御文字コマンド

ユーザは特殊なキーを押すことにより，CP／M-86に対し，特定の機能を実行するように指示することができる．制御文字コマンドを使用すれば，CP／M-86に対し，画面移動（スクローリング）の開始および停止，現在の動作の中断，あるいはスクリーンの内容のプリンタへのエコーを指示することができる．下記の表は制御文字コマンドをまとめたものである．

表3-3．制御文字コマンド

コマンド	意味
CTRL-C	現在実行中のプログラムを終了する．あるいはシステム・プロンプトの後でタイプすると，システムおよび省略時ドライブを初期化し，すべてのドライブをRWステータスにセットする．
CTRL-P	コンソール上の すべての動作をプリンタにエコーする． もう一度CTRL-Pを入力すると，プリンタへのエコーが終了する．このコマンドが機能するのはシステムがプリンタに接続されている場合だけである．
CTRL-S	画面移動（スクローリング）を開始したり終了したりする．表示が非常に速くスクローリングしているために，内容を読み取れない場合には，CTRL-Sを押す．他のキーあるいはCTRL-Sをもう一度押すと，スクリーンは再びスクローリングを開始する．

第4章
コマンドのまとめ

この章では，コマンド行に含まれるファイル記述の各部分の表わし方について説明する．またコマンド行の省略可能な部分を示すための表記法や，その他の構文表記法を示す．この章の残りの部分は，標準的なCP／M-86のコマンドを参照するための資料として使用することができる．

組込みコマンドとトランジェント・ユーティリティ・コマンドとが，アルファベット順に混合して示されている．各コマンドの後にその動作についての簡単な説明と例が示されている．さらに複雑なコマンドについては後で詳しく説明する．例えばEDは第5章で説明されているが，ASM-86，DDT-86，GENCMDはCP／M-86システム・ガイドに示されている．

4.1 コマンドの各部分の名称の統一

すでに説明したように，ファイル記述はいくつかの部分によって構成されている．混乱を避けるために，コマンド行の説明で使用する各部分の正式な名前を示しておく．ファイル記述の3つの部分は次の通りである．

・ **ドライブ記述子**——参照するファイルまたはファイル・グループが含まれているディスク・ドライブA，B，CまたはDであり，これは必ずしも指定する必要がない．ドライブ記述子がコマンド行に指定されている場合には，その後にコロンを入力しなければならない．

・ **ファイル名**——ファイルまたはファイル・グループの名前であり，1文字から8文字の長さである．

・ **ファイル型**——ファイルまたはファイル・グループのファミリ名であり，長さは1文字から3文字で，必ずしも指定する必要はない．ファイル型が存在する場合には，ファイル名との間をピリオドで区切らなければならない．

ファイル記述の一般形式は次の形式である．

```
d:filename.typ
```

上記の形式で〝d：〟は任意のドライブ記述子を示し，〝filename〟は1文字から8文字のファイル名を示し，〝.typ〟は1文字から3文字の任意のファイル型を示す．

CP／M-86のファイル記述の各要素の正しい組合わせを示すと次のようになる．

- filename
- d:filename
- filename.typ
- d:filename.typ

ドライブ記述子を指定しない場合，CP／M-86は自動的に省略時ドライブを割当てる．ピリオドとファイル型を省略した場合には，CP／M-86は自動的に3つのブランクから構成されるファイル型を割当てる．

この一般形式を〝ファイル記述〟と呼ぶ．ファイル記述は，ドライブ記述子で指定されているオンライン・ディスクのディレクトリに含まれる特定のファイル，またはファイル・グループに名前を付ける．例えば次のファイル記述は，

```
B:MYFILE.A86
```

ドライブが〝B：〟であり，ファイル名が〝MYFILE〟，ファイル型が〝A86〟であることを示す．コマンド構文の説明では〝ファイル記述〟を次のように省略して書くことにする．

```
filespec
```

CP／M-86のコマンドの中には，コマンド・テイルのファイル名およびファイル型の部分にワイルド・カードを指定できるものがある，例えば次のファイル記述は，

```
B:MY*.A??
```

ドライブ記述子が〝B：〞であり，ファイル名が〝MY＊〞，ファイル型が〝A？？〞であることを示す．このファイル記述はディレクトリの中に含まれる複数のファイルと対応する．

これでコマンド・キーワード，コマンド・テイル，制御文字，省略時ドライブ，オンライン・ドライブおよびワイルド・カードについて理解できたと思う．またファイル記述（filespec），ドライブ記述子（drive specifier），ファイル名(filename)，ファイル型（filetype）という正式な名称の使い方についても説明した．これらの概念は完全なコマンド行を作成するのに必要な背景となるものである．

4.2 コマンドの説明方法

この章では組込みコマンドおよびトランジェント・ユーティリティ・コマンドをアルファベット順に示す．各コマンドの説明は次のような形式で行なわれる．

・ まずコマンド・キーワードが大文字で示される．さらに必要に応じて，次のコマンドの目的をよりよく表す英文のフレーズが，キーワードの後にカッコで囲んで示される．

・ "構文" の部分では，コマンド行を構成する１つまたは複数の一般形が示される．

・ "タイプ" の項ではキーワードが組込みコマンドなのか，トランジェント・ユーティリティ・コマンドなのかが示される．組込みコマンドは常に使用することができるが，トランジェント・ユーティリティ・コマンドは，ＣＭＤプログラム・ファイルとしてオンライン・ディスク上に存在しなければならない．

・ "目的" の項では，コマンド・キーワードの一般的な使い方を定義する．

・ "備考" の項では，例外事項および特殊事項を示す．

・ "例" の項では，コマンド・キーワードを使った正しいコマンド行が示される．ユーザとオペレーティング・システムの対話の例をわかりやすくするために，ユーザが入力した文字は太文字で示される．ＣＰ／M-86の応答は普通の文字で示される．

構文行の表記方法では，次の規則を使って一般的なコマンドの形式を説明する．

・大文字の単語は，ユーザが示されている通りの綴りでタイプしなければならないものであるが，実際にタイプする場合には大文字と小文字をどのように組合わせてもかまわない．

・ イタリック体で示されている小文字の単語は一般的な意味であり，そのコマンドに対して本文の中でさらに定義が行なわれる．例えば "オプション (option)" という単語があれば与えられたオプション・リストの中から選択することができる．

・ ｎの代りに数字を使うことができる．

・ "d：", "ファイル名 (filename)", ".typ", "ファイルスペック (filespec)" には前の章で説明した一般的な意味がある．

・ 特に指定されていない限り，スペースが示されている場所には，１つまたは複数のスペース文字を指定しなければならない．例えばＰＩＰオプションの場合，スペースで各オプションを区切る必要はない．

・ ｛｝に囲まれている項目は，省略可能である．従ってコマンドを入力する場合，そのような省略可能な項目は必ずしも指定する必要がない．省略可能な項目は，コマンド行に特定の効果を追加するものである．

・ （・・・）は，前の項目が何度も繰返されることを示す．

・ 構文行に１つまたは複数の代替項目を入力できる場合には，縦線（｜）によってそれぞれの代替項目が区切られる．この縦線は "または" を示すと考えることができる．

・ ↑またはＣＴＲＬはキーボードからコントロール・キーを入力することを示す．

・ 他の句読点はすべてコマンド行の中に含まなければならない．

構文表記法の例をいくつか示すことにしよう．ＣＰ／M-86のトランジェント・ユーティリティ・コマンドＳＴＡＴ（ステータス）は，すべてのオンライン・ドライブの使用可能な空間の大きさをキロバイト単位で表記する．またそれぞれのファイルが使用している空間の大きさも，キロバイト単位で表示する．

ＳＴＡＴはさらに，読取り専用（ＲＯ）または読み書き（RW），システム（ＳＹＳ）またはディレクトリ（ＤＩＲ）属性をファイルに割当てることができる．

ＳＴＡＴコマンドの構文の項では，コマンド行の構文表記法がどのように使用されるかが示されている．

構文：
```
STAT { filespec   {RO | RW | DIR | SYS } }
     |            |                    | |
     |            -----------optional------ |
   --------------- optional -------------
```

これは，コマンド・キーワードＳＴＡＴの後にあるコマンド・テイルが省略可能であることを示している．ＳＴＡＴだけでも有効なコマンドであるが，コマンド行の中にファイル記述を含むこともできる．従ってＳＴＡＴ　filespec も正しいコマンドである．さらにファイル記述の後に別の省略可能な値を指定することもできる．その省略可能な値とは次のいずれかである．

```
RO
RW
DIR
SYS
```

従って

```
STAT filespec RO
```

上記のコマンドは有効なコマンドである．

第3章で，ファイル名やファイル型の中に万能札を含むことができることを示した．ＳＴＡＴコマンドではファイル記述にワイルド・カードを指定することもできる．

この構文を使用すれば，何通りもの正しいコマンド行を構成することができる．

```
STAT
STAT X.A86
STAT X.A86 RO
STAT X.A86 SYS
STAT *.A86
STAT *.* RW
STAT X.* DIR
```

ＣＰ／M-86のコマンドＰＩＰ（周辺装置交換プログラム）は，ファイルをコピーするプログラムである．ＰＩＰはスクリーンに表示されている情報を，ディスクやプリンタにコピーすることができる．またＰＩＰは複数のファイルを１つのより大きなファイルに結合することができる．さらにＰＩＰはコピーを行なった後で，そのファイルに別の名前を付けることができる．コマンド行の表記法の使い方の例として，ＰＩＰコマンド行の１つの形式をとりあげることにしよう．

構文：
```
PIP dest-filespec=source-filespec{,filespec...}
```

この例で，dest-filespecはデータを受取る目的ファイル記述，または周辺装置（例えばプリンタ）としてさらに定義されている．同様にsource-filespecはデータを送るファイル記述，または周辺装置（例えばキーボード）である．ＰＩＰではファイル名およびファイル型の中にワイルド・カードを指定することができる．（ＰＩＰの他の機能についてはＰＩＰコマンドのまとめを参照されたい）．従って他の構文から多くの正しいコマンド行を作成することができる．その一部を下記に示す．

```
PIP NEWFILE.DAT = OLDFILE.DAT
PIP B: = A:THISFILE.DAT
PIP B:X.BAS = Y.BAS, Z.BAS
PIP X.BAS = A.BAS, B.BAS, C.BAS
PIP B: = A:*.BAK
PIP B: = A:*.*
```

4.3 ＡＳＭ-86（アセンブラ）コマンド

構文：

```
ASM86 filespec { $parameter-list }
```

タイプ：

トランジェント・ユーティリティ．

目的：

ＡＳＭ-86ユーティリティは，8088および8086アセンブリ言語ソース・ステートメントを機械コード形式に交換する．

ＡＳＭ-86アセンブラの動作については，ＣＰ／Ｍ-86プログラミングガイドに詳しく説明されている．

備考：

ファイル記述は，交換したい8086アセンブリ言語プログラムを記憶している文字ファイルの名前を指定する．ファイル型を省略した場合には，Ａ86とみなされる．コマンド・テイルの中に省略値を無効にするパラメータが含まれていない限り，アセンブラは出力ファイルに対する目的ドライブとして，ファイル記述のドライブ記述子を使用する．

アセンブラが作成する３つの出力ファイルには，次のファイル型が与えられる．

ＬＳＴ　注釈付きのソース・リストを含む．
Ｈ８６　"16進"8086機械コードを含む．
ＳＹＭ　プログラマが定義したシンボルとそれぞれのプログラム相対アドレスを含む．

アセンブラはＬＳＴ，Ｈ86およびＳＹＭファイルに対して，ソース・ファイル名と同じファイル名を割当てる．

パラメータ・リストの中の省略可能なパラメータを指定することにより，アセンブリの処理を制御することができる．各パラメータは１文字のパラメータであり，その後に１文字の装置名が続く．パラメータはブランクで区切ることができるが，各パラメータ文字のすぐ後に装置名を指定しなければならない．

パラメータ文字はＡ，Ｈ，Ｐ，Ｓ，Ｆである．装置名はＡからＰまでの文字であり，それぞれドライブ文字に対応している．装置名として文字Ｘ，Ｙ，Ｚを使用する場合には，それぞれ特殊な意味を持つ．

Ｘ　はスクリーンである．
Ｙ　はプリンタである．
Ｚ　はゼロ出力である．

Ａというパラメータ文字を使用すれば，省略時ドライブ記述子を無効にしてソース・ファイルを入手することができる．正しいパラメータはＡＡからＡＰ迄である．

Ｈというパラメータ文字を使用すれば，省略時ドライブ記述子を無効にしてＨ86ファイルを受取ることができる．正しいパラメータはＨＡからＡＰ，およびＨＸ，ＨＹ，ＨＺである．

Ｐというパラメータ文字を使用すれば，省略時ドライブ記述子を無効にしてＬＳＴファイルを受取ることができる．正しいパラメータはＰＡからＰＰ，およびＰＸ，ＰＹ，ＰＺである．

Ｓというパラメータ文字を使用すれば，省略時ドライブ記述子を無効にしてＳＹＭファイルを受取ることができる．正しいパラメータはＳＡからＳＰ，およびＳＸ，ＳＹ，ＳＺである．

Fというパラメータ文字を使用すれば，"16進" 出力ファイルの形式を選択することができる．正しいパラメータはＦＩとＦＤである．パラメータＦＩはIntel形式の "16進" ファイル出力を選択し，パラメータＦＤはDigital Researchの形式の"16進" ファイル出力を選択することができる．パラメータとしてＦＩもＦＤも指定されていない場合には，ＦＤであるとみなされる．Intelのコンパイラまたはアセンブラによって生成されたプログラムと組合せる場合には，ＦＩを使用する．

コマンド行に矛循するパラメータが指定されている場合には，右側のパラメータが有効である．

例：

```
A>ASM86 X
```

ＡＳM86．ＣＭＤというファイルがドライブAに存在しなければならない．ソース・ファイルX．A86がドライブＡから読取られ，X．ＬＳＴ，X．H86，X．ＳＹＭがドライブAに書込まれる．

```
B>ASM86 X.ASM $PX
```

ＡＳM86．ＣＭＤというファイルがドライブBに存在しなければならない．ソース・ファイルX．ＡＳＭがドライブＢから読取られる．リストがスクリーン上に表示され，X．H86およびX．ＳＹＭというファイルがドライブＢに書込まれる．

```
A>ASM86 B:MYPROG $PY HC
```

ソース・ファイルMＹＰＲＯＧ．A86がドライブＢから読取られ，リストがプリンタに出力され，ファイルMＹＰＲＯＧ．H86がドライブCにファイルMＹＰＲＯＧ．ＳＹＭがドライブＢに書込まれる．

```
A>B:ASM86 X $SZ
```

ＡＳM86．ＣＭＤというファイルがドライブBに存在しなければならない．X．A86というファイルがドライブＡから読取られる．X．ＬＳＴおよびX．H86というファイルがドライブAに書込まれる．X．ＳＹＭというファイルは生成されない．

4.4 COPYDISK（コピー・ディスク）コマンド

構文：

COPYDISK

タイプ：

トランジェント・ユーティリティ．

目的：

COPYDISKユーティリティは，ディスク上のすべての情報を別のディスクにコピーする．CP／M-86システム・トラックがソース・ディスクに存在する場合，そのトラックにコピーされる．

新しいディスクにコピーする場合には，まずディスク・フォーマティング・プログラムを使って，そのディスクをフォーマティングしておかなければならない．使用済のディスクコピーする場合には，COPYDISKはブランクも含めてソース・ディスク上のすべての情報を目的ディスクの情報の上から重ねて書込む．

備考：

COPYDISKの使い方に関する指示を表示するには，キーワードHELPとコマンド・テイルCOPYDISKを入力する．

あるディスクから別のディスクへのコピーを成功させるには，目的ディスクが書込み保護されていないことを確認する必要がある．したがって，目的ドライブにディスクを挿入する前に，ディスクの書込み保護ノッチがファイル・タブによってカバーされていることを調べておかなければならない．

COPYDISKは1トラックずつ，1セクタずつ正確にコピーするユーティリティであり，ディスク全体をコピーする方法としては最も速い方法である．しかしソース・ディスク上に多くのファイルが作成され，消去されている場合には，それぞれのファイルに属するレコードが，ディスク上のいろいろな場所にランダムに配置されていることがある．この場合には，PIPを使ってファイルをコピーし，すべてのレコードを新しいディスクに順に出力する方がその後の処理の効率を高めることができるだろう（コピー速度は遅いが）．

例：　A>**COPYDISK**

COPYDISKを呼出すと，ソース・ディスクおよび目的ディスクを指示するプロンプトが出力される．1つのドライブ・システムだけしかなく，ドライブBがない場合でも，この形式を使用する．この場合，A：とB：はドライブ名でなくディスク名として考えられる。COPYDISKはディスクを交換しなければならない時がきたら，スクリーン上のステータス行にその指示を出力する．

次の例は，COPYDISKがマスタ・ディスク（ディスクA：）から，新しいディスク（ディスクB：）へコピーする例を示したものである．COPYDISKを呼び出すと，例の最初の行に示されている情報が表示される．

```
CP/M-86 Full Disk Copy Utility
     Version 2.0

Enter Source Disk Drive (A-D) ? A

Destination Disk Drive (A-D) ? B

Copying disk A: to disk B:
Is this what you want to do (Y/N) ?  Y
Copy started
Reading track nn     (After read, new text appears)
Writing track nn     (After write, next message is)
Verifying track nn
Copy completed.

Copy another disk (Y/N) ? N
Copy program exiting

A>
```

4.5 DDT-86（ダイナミック・デバッキング・ツール）コマンド

構文：

```
DDT86 { filespec }
```

タイプ：

トランジェント・ユーティリティ

目的：

DDT-86ユーティリティを使用すれば，8086および8088プロセッサのために開発されたプログラムを監視し，テストすることができる．

DDT-86の1文字のコマンド，そのパラメータおよびオプションが表4-1に示されている．実際のコマンド文字は大文字で示されている．パラメータは小文字で，コマンド文字の後に示されている．省略可能なオプション項目は，中カッコで囲まれている．引数は，表4-1の後に示されているリストの中の適切な値と置換えることができる．

表4-1．DDT-86コマンド

コマンド	意味
As	（アセンブル）アセンブリ言語ステートメントを入力する．
Bs, f, sl	（ブロック比較）メモリのブロックを比較する．
D {W} {s {, f} }	（表示）メモリを16進数およびASCIIで表示する．
Efilespec	（実行）プログラムを実行するためにロードする．
Fs, f, bc	（充てん）メモリ・ブロックを充てんする-バイト．
FWs, f, wc	（充てん）メモリ・ブロックを充てんする-ワード．
G {s} {, b1 {, b2} }	（Go）実行を開始する．
Hwc1, wc2	（16進数）16進数の和および差
Icommand tail	（入力）入力コマンド行をセット・アップする
L {s {, f} }	（リスト）メモリをニモニック形式でリストする
Ms, f, d	（移動）メモリ・ブロックを移動する
Rfilespec	（読取り）ディスク・ファイルをメモリに読取る
S {W} s	（セット）メモリに値をセットする
T {n}	（トレース）プログラムの実行をトレースする
TS {n}	（トレース）すべてのレジスタをトレースし示す
U {n}	（非トレース）トレースを行なわずに実行を監視する
US {n}	（非トレース）すべてのレジスタを監視し，示す
V	（検証）ディスクを読取った後，メモリ・レイアウトを示す
Wfilespec {, s, f}	（書込み）ブロックの内容をディスクに書込む
X {r}	（検査）CPUレジスタを調べ変更する

パラメータ	置換える値
bc	バイト定数
b1	ブレークポイント1
b2	ブレークポイント2
d	データの宛先
f	最終アドレス
n	実行したい命令の数
r	レジスタまたはフラグ名
S	開始アドレス
S1	2番目の開始アドレス
W	16ビットのワード
WC	ワード定数

DDT-86の動作および各1文字のコマンドについては，CP／M-86 プログラミングガイドに詳しく説明されている．

備考：

ファイル記述が存されていない場合，DDT-86はユーザ・メモリにロードされ，テスト・プログラムはロードされない．最初にテスト・プログラムをロードするまで，DDT-86のコマンドG，T，Uを使用してはならない．テスト・プログラムは通常コマンドEを使ってロードされる．

ファイル記述が指定されている場合，DDT-86とファイル記述によって指定されているテスト・プログラム・ファイルの両方がユーザ・メモリにロードされる．DDT-86の監視のもとでテスト・プログラムの実行を開始するにはG，T，Uのいずれかを使用する．

ファイル記述からファイル型が省略されている場合には，ファイル型はCMDであるとみなされる．

DDT-86は16進（H86）形式で直接にテスト・プログラムをロードできない．従って，まずGENCMDユーティリティを使って，コマンド・ファイル形式（CMD）に変換しておかなければならない．

例：　**A>DDT86**

DDT-86ユーティリティが，ドライブAからユーザ・メモリにロードされる．DDT-86はコマンドを受取れる状態になると〝－〞というプロンプトを表示する．

A>B:DDT86 TEST.CMD

DDT-86ユーティリティがドライブBからユーザ・メモリにロードされる．次に，TEST．CMDというプログラム・ファイルがドライブAからユーザ・メモリにロードされる．DDT-86はファイルがロードされたアドレスを表示し〝－〞というプロンプトを出力する．

4.6 DIR（ディレクトリ）組込みコマンド

構文：

```
DIR  {d:}
DIR  {filespec}

DIRS {d:}
DIRS {filespec}
```

タイプ：

組込みコマンド

目的：

DIRおよびDIRS組込みコマンドは，オンライン・ディスクのディレクトリにカタログされているファイル名を表示する．DIR組込みコマンドを使用すれば，現在のユーザ番号のもとでディレクトリ（DIR）属性を持つファイルの名前を表示する．DIRではファイル記述の中にワイルド・カードを指定することができる．

DIRSコマンドは，現在のユーザ番号のもとでシステム（SYS）属性を持つファイルの名前を表示する．従って，同じドライブの他のユーザ番号からユーザ番号0で記憶されたシステム（SYS）ファイルをアクセスできる場合でも，DIRSは現在のユーザ番号が0ならユーザ0のファイルの名前を表示する．DIRSでは，ファイル記述の中にワイルド・カードを指定することができる．

備考：

ドライブおよびファイル記述が省略されている場合には，DIRコマンドは，現在のユーザ番号のもとで省略時ドライブにセットされているディスクのDIR属性を持つすべてのファイルの名前を表示する．同様に，DIRSはそのようなSYSファイルを表示する．

ドライブ記述子だけが指定されており，ファイル名とファイル型が省略されている場合には，DIRコマンドは現在のユーザ番号のもとで指定されているディスクに格納されているすべてのDIRファイルの名前を表示する．同様にDIRSは，そのようなSYSファイルを表示する．

ファイル記述にワイルド・カード文字が含まれている場合，それに対応するすべてのファイル名がスクリーンに表示される．

ファイル名がファイル記述と対応しない場合や，指定されているドライブのディスクのディレクトリのようなファイルがカタログされていない場合には，DIRコマンドは次のメッセージを表示する．

```
NO FILE
```

システム（SYS）ファイルが，指定されているドライブに存在する場合，DIRは次のメッセージを表示する．

```
SYSTEM FILE(S) EXIST
```

システム以外のファイル（DIR）が，指定されているドライブに存在する場合，DIRSは次のメッセージを表示する．

```
NON-SYSTEM FILES(S) EXIST
```

ドライブ記述子の中にワイルド・カード文字を指定することはできない．

例：　A>DIR

現在のユーザ番号で，ドライブAにセットされているディスクのディレクトリにカタログされているすべてのDIRファイルがスクリーンに表示される．

A>DIR B:

現在のユーザ番号で，ドライブBのディスクにカタログされているすべてのDIRファイルがスクリーンに表示される．

A>DIR B:X.A86

ファイルX．A86がドライブBにセットされているディスクに存在する場合，DIRコマンドはスクリーンに名前X．A86を表示する．

A>DIR *.BAS

現在のユーザ番号で，ドライブAにセットされているディスクのファイル型BASを持つすべてのDIRファイルがスクリーンに表示される．

B>DIR A:X*.C?D

現在のユーザ番号で，ドライブAのディスクにカタログされているファイルで，ファイル名が文字Xから始まり，3文字のファイル型の1文字目がCで，3文字目がDであるすべてのDIRファイルがスクリーンに表示される．

A>DIRS

現在のユーザ番号で，ドライブAのディスクに含まれるファイルで，システム（SYS）属性を持つすべてのファイルが表示される．

A>DIRS *.CMD

現在のユージ番号で，ドライブAにセットされているディスクのファイルで，システム（SYS）属性を持ち，ファイル型がCMDであるすべてのファイルが表示される．

4.7 ED（テキスト・エディタ）コマンド

構文：

```
ED input-filespec {d: | output-filespec}
```

タイプ：

トランジェント・ユーティリティ

目的：

EDユーティリティを使用すれば，ディスク・ファイルを生成したり，編集したりすることができる．

EDユーティリティは，〝行〟および〝文脈〟の編集を行なうエディタである．つまり文字ファイルを1行ずつ作成したり変更したりすることもできるし，1行の中の1文字ずつを参照することもできる．
EDユーティリティを使用すれば，ファイル記述で指定されたファイルの作成および変更が可能である．

EDユーティリティでは，ユーザ・メモリの一部をアクティブなテキスト〝バッファ〟として使用する．このバッファの中でファイルに含まれる文字を追加したり，削除したり，変更したりすることができる．Aコマンドを使用すれば，ファイルの一部または全部をバッファに読み取ることができる．WまたはEコマンドを使用すれば，文字の一部または全部をバッファからファイルに書込むことができる．

CPと呼ばれる見かけ上の〝文字ポインタ〟は，バッファの先頭，バッファに含まれる2文字の間，あるいはバッファの終りを示す．

ユーザは〝コマンド〟または〝挿入〟モードで，EDユーティリティと対話することができる．EDがコマンド・モードの場合には，表示画面に〝*〟というプロントが表示される．〝*〟が表示されると，1文字のコマンドを入力することができる．そのコマンドはテキストをバッファから読取ったり，CPを移動したり，あるいはEDの操作モードを変更したりする．

表4-2．EDコマンド

コマンド　　　　　　　　　　**動　　作**

nA

オリジナル・ファイルからメモリ・バッファへn行を追加する

0A

バッファの半分が一杯になるまでファイルを追加する

#A

バッファが一杯になるまで（またはファイルの終りに到達するまで）ファイルを追加する

B, -B

CPをバッファの先頭（B）または終り（－B）へ移動する

nD, －nD

CPの前の（－D）n文字またはCPの後の（D）n文字を削除する

E

新しいファイルをセーブし，ＣＰ／M-86に戻る

Fstring{↑z}

文字ストリングを見つける

H

新しいファイルをセーブし，新しいファイルをオリジナル・ファイルとして再び編集する

I

挿入モードに入る．挿入モードから抜け出すには↑Ｚを使う

Istring{↑Z}

ＣＰの位置にストリングを挿入する

Jsearch_str^Zins_str^Zdel_to_str{↑Z}

ストリングを並置する

nK, −nK

ＣＰの位置からｎ行を削除する

nL, −nL, 0L

ＣＰをｎ行だけ移動する

nMcommands

コマンドをｎ回実行する

n, −n

ＣＰをｎ行だけ移動し，その行を表示する

n：

行ｎへ移動する

：ncommand

コマンドを行ｎから実行する

Nstring{↑Z}

ストリング探索の拡張

O

オリジナル・ファイルに戻る

nP,−nP

ＣＰを23行だけ前進させ，コンソールに23行を表示する

Q

新しいファイルを放棄し，ＣＰ／M-86に戻る

R

X＄＄＄＄＄＄＄．LIBというファイルをバッファに読み取る

Rfilespec {↑Z}

ファイル記述で示したファイルをバッファに読取る

Sdelete string^Zinsert string {↑Z}

ストリングを置換える

nT, −nT, 0T

n行をタイプする

U, −U

大文字への変換

V, −V, 0V

行への番号付けをnoまたはoffにし，使用されていないバッファ空間を表示する

nW

n行を新しいファイルに書込む

nX

n行をX＄＄＄＄＄＄＄．LBIに書込む，または追加する

nXfilespec {↑Z}

n行をファイル記述で示したファイルに書込む.または前のXコマンドが同じファイルに適用される場合には，n行をファイルに追加する．

0X

ファイルX＄＄＄＄＄＄＄．LIBを削除する

0Xfilespec {↑Z}

ファイル記述で示したファイルを削除する

nZ

n秒間だけ待つ

EDユーティリティの動作および各コマンドの使い方については，第5章に詳しく説明されている．

備考：

2番目のファイル記述を指定するのは，最初のファイル記述で指定されているファイルがすでに存在し，オリジナル・ファイルをそのまま残しておきたい場合だけである．2番目のファイル記述によって指定されるファイルは，最初のファイルから変更されたテキストを受取る．この場合，最初のファイルは変更されない．

2番目のファイル記述にドライブ記述子だけしか指定されていない場合には，2番目のファイル名およびファイル型は，最初のファイル名およびファイル型と同じものとなる．

最初のファイル記述によって指定されているファイルが存在しない場合，ＥＤユーティリティはそのファイルを作成し，次のメッセージを出力する．

```
NEW FILE
```

２番目のファイル記述が省略されている場合には，置換する前に，オリジナル・ファイルのファイル型をＢＡＫに変更することにより，オリジナル・ファイルを保存することができる．ファイル型ＢＡＫのファイル記述を指定したＥＤコマンド行を入力すると，ＥＤはＢＡＫファイルの新しく編集されたバージョンを作成しそれをセーブするが，ソース・ファイルを削除するので，バック・アップは残されない．オリジナルのＢＡＫファイルをセーブしたい場合には，まずＲＥＮコマンドを使ってファイル型をＢＡＫから変更し，ＥＤがそれをＢＡＫに変更できるようにしておかなければならない．

２番目のファイル記述を指定し，それに最初のファイル記述と同じ名前を与えた場合，ＥＤは出力ファイルの新しく編集されたバージョンを作成しセーブするが，出力ファイルと同じ名前のファイルが存在しないように，オリジナル入力ファイル記述を削除する必要が生じる．勿論，同じユーザ番号のもとで，同じドライブに同じ名前のファイルを２つ持つことはできない．

最初のファイル記述によって指定されたファイルがすでに存在する場合には，Ａコマンドを入力し，そのファイルをバッファに読込まなければならない．ファイルの大きさがバッファの大きさより小さい場合には，次のコマンドを入力すれば，

```
#a
```

ファイル全体がバッファに読込まれる．

ｉ（挿入）コマンドは，ＥＤユーティリティを挿入モードにする．このモードでは，ユーザがタイプした文字は，現在のＣＰの位置から順にバッファに記憶される．

挿入モードでタイプされた１文字のコマンドは，コマンドとしては解釈されずバッファに記憶されるだけである．挿入モードからコマンド・モードに戻るには，ＣＴＲＬ-Ｚをタイプする．

１文字のコマンドは、通常小文字でタイプされる．コマンドの後に他のコマンド文字が続く場合には，そのコマンドを入力した後でＣＴＲＬ-Ｚを入力しなければならない．

大文字で１文字のコマンドを入力すると，ＣＴＲＬ-Ｚが入力されるまで，すべての文字は内部的に大文字に変換される．ＣＴＲＬ-Ｚが入力されるとコマンドは終了する．

許可されている場合には，行番号がスクリーンの左側に表示される．行番号の形式は次の通りである．

```
nnnnn:
```

ｎｎｎｎｎは，１から65535までの数値である．行番号は参照しやすいように表示されるものであり，バッファや文字ファイルには含まれない．

ＣＰがバッファの先頭または終りにある場合には，スクリーン上の行は次の文字から始まる．

```
:
```

例：

```
A>ED MYPROG.A86
```

このようなファイルが存在しない場合には，このコマンド行はファイルＭＹＰＲＯＧ．A86をドライブAに作成する．この場合下記のコマンド・プロンプトがスクリーンに表示される．

```
:*
```

このプロンプトは，ＣＰがバッファの先頭にあることを示す．ファイルが既に存在する場合には，次のコマンドを入力し，

```
:*#a
```

バッファを充てんする．次に下記のコマンドをタイプし，

```
:*0p
```

バッファの最初の23行をスクリーンに表示する．文字ファイルの変更が終了したら次のコマンドを入力し，

```
:*e
```

ＥＤユーティリティを終了する．ＥＤユーティリティはオリジナル・ファイルを変更せず，ＭＹＰＲＯＧ．ＢＡＫとして保存し，変更後のファイルをＭＹＰＲＯＧ．A86として保存する．

```
A>ED MYPROG.A86 B:NEWPROG.A86
```

オリジナル・ファイルはＭＹＰＲＯＧ．A86であり，省略時ドライブAに存在する．ＥＤユーティリティが終了したとき，オリジナル・ファイルは変更されないままの状態であり，変更後のファイルはＮＥＷＰＲＯＧ．A86としてドライブBに記憶される．

```
A>B:ED MYPROG.A86 B:
```

ED．CMDファイルは，ドライブBに存在しなければならない．オリジナル・ファイルはＭＹＰＲＯＧ．A86であり，ドライブAに存在する．オリジナル・ファイルは変更されず，変更済のプログラムはＭＹＰＲＯＧ．A86としてドライブBに記憶される．

4.8 ＥＲＡ（削除）組込みコマンド

構文：

```
ERA filespec
```

タイプ：

組込みコマンド

目的：

ＥＲＡ組込みコマンドは，１つまたは複数のファイルをディスクのディレクトリから削除する．コマンド・テイルの中にワイルド・カード文字を指定することができる．削除されたディレクトリおよびデータ空間は自動的に開放されるので，別のファイルで使用できる．

備考：

ＥＲＡコマンドを使用する場合には，ファイル記述を満足するすべてのファイルがディスク・ディレクトリから削除されるので，十分に注意を払う必要がある．

コマンド行の形式は下記の通りである．

```
ERA {d:}*.*
```

このようにすべてのファイル空間を開放する場合，ユーザの承認が必要である．従って次のメッセージが出力される．

```
All (Y/N)?
```

すべてのファイルを削除したい場合には〝y〟を入力し，ファイルの削除を行ないたくない場合には〝n〟を入れる．

ファイル記述と対応するファイルが存在しない場合には，スクリーンに次のメッセージが出力される．

```
NO FILE
```

例：

```
A>ERA X.A86
```

このコマンドは，ファイルX．A86をドライブAにセットされているディスクから削除する．

```
A>ERA *.PRN
```

ファイル型がＰＲＮであるすべてのファイルが，ドライブAにセットされているディスクから削除される．

```
B>ERA A:MY*.*
```

ＭＹで始まるファイル名を持つ各ファイルが，ドライブAにセットされているディスクから削除される．

```
A>ERA B:*.*
```

ドライブBのすべてのファイルがディスクから削除される．この操作ではＥＲＡコマンドが次のメッセージを出力するので〝y〟を応答しなければならない．

```
All (Y/N)?
```

4.9 GENCMD (CMDファイルの生成) コマンド

構文:

```
GENCMD filespec {8080 CODE[An,Bn,Mn,Xn] DATA[An,Bn,Mn,Xn]
            STACK[An,Bn,Mn,Xn] EXTRA[An,Bn,Mn,Xn] X1[...}
```

タイプ:

トランジェント・ユーティリティ

目的:

GENCMDユーティリティは，ASN-86およびその他の言語プロセッセの16進出力を使って，CMDファイルを作成する．ファイル記述の後に，省略可能なパラメータ・リストを指定することができる．

トランジェント・ユーティリティ・コマンドとなるアセンブリ言語プログラムを書く場合には，GENCMDの使い方を知っておかなければならない．

GENCMDの動作については，CP/M-86システムガイドに詳しく説明されている．

パラメータ・リストは，最大9つのキーワードと対応する値のリストから構成される．キーワードは次の通りである．

```
8080   CODE   DATA   STACK   EXTRA   X1   X2   X3   X4
```

キーワード8080は，CMDファイルを〝8080メモリ・モデル〟として識別する．8080メモリ・モデルでは，コードおよびデータ・グループがオーバラップする．他のキーワードは，メモリに対する特定の必要条件を持つセグメント・グループを定義する．メモリに対する必要条件を定義する値はカンマによって区切られ，([])で囲まれて各キーワードの後に指定される．角括弧で囲まれたキーワードと関連する値は，他のキーワードと少なくとも1つのブランクで区切らなければならない．

角括弧に囲まれる値は，下記のように定義される．但しnは1桁から4桁までの16進定数である．値nは〝パラグ値を表わす．各パラグラフは16バイトの長さである．パラグラフの値はn＊16のバイト値，つまり16進数ではhhhh0に対応する．

- An　グループを絶対ロケーションする
- Bn　グループを16進ファイルのアドレスnから始める
- Mn　グループは少なくともn＊16バイトを必要とする
- Xn　グループは最大n＊16バイトのアドレスまで指定することができる

備考:

8ビット・マイクロプロセッサからCP/M-86に変換されるプログラムの場合には，8080キーワードを使用する．この場合プログラムはある領域にロードされ，コードとデータ・セグメントがオーバラップする．プログラムの中のコード・セグメントは，ロケーション100Hから始まらなければならない．

メモリの絶対ロケーションにロードしなければならないグループに対しては，Anを使用する．プログラムを実行するとき，要求される絶対領域が使用可能であるかどうかわからない場合，コマンド・テイルの中でAnを使用してはならない．

入力する16進ファイルの中にセグメント・グループを識別する情報が含まれていない場合には，Bnを使用する．ASM-86のパラメータFIが含まれていない限り，Digital Resarch社のASM-86アセンブラから，H86ファイルが出力される場合には，この値は必要ない．

セグメントの終りに初期化されていないデータ領域を含むデータ・セグメントが存在する場合には，Mnを使用する．

Mnによって与えられている最小値よりも大きなデータ領域を使用できる場合には，Xnを使用する．

例：

```
A>GENCMD MYFILE
```

ファイルMYPROG．H86がドライブAから読取られる．出力ファイルMYPROG．CMDは，ドライブAに書込まれる．入力ファイルH86にはそのプログラムが特定のメモリ・モデルで動作することを示す情報が含まれる．

```
B>GENCMD MYFILE  CODE[A40]  DATA[M30,XFFF]
```

ファイルMYFILE．H86がドライブBから読取られる．MYFILE．CMDという出力ファイルがドライブBに書込まれる．コード・グループは16進数で400というロケーションにロードしなければならない．データ・グループには少なくとも16進数で300バイトのメモリ空間が必要であるが，可能な場合にはプログラムは最大FFF0バイトまで使用することができる．

4.10 HELP（ヘルプ）コマンド

構文：

```
HELP {topic} {subtopic1 subtopic2 ... subtopic8}{[P]}
```

タイプ：

トランジェント・ユーティリティ

目的：

ＨＥＬＰコマンドを使用すれば，本書に説明されているすべてのＣＰ／Ｍ-86 コマンドの要約された情報を入手することができる．コマンド・テイルの指定されていないＨＥＬＰを入力すると，すべての項目のリストが表示される．コマンド・テイルにトピックが指定されたＨＥＬＰを指定すると，そのトピックに関する情報とサブトピックの情報が表示される．トピックとサブトピックの指定されたＨＥＬＰを入力すると，特定のサブトピックに関する情報が表示される．

備考：

ＨＬＥＰは指定されたトピックの情報を表示した後，スクリーンに特殊なプロンプトＨＥＬＰ＞を表示する．このプロンプトが表示されたら他のトピックを指定し，追加情報を入手することもできるし，単にＲＥＴＵＲＮキーを押せば，ＣＰ／Ｍ-86 システム・プロンプトに戻ることができる．

トピックやサブトピックの名前は短縮することができる．通常それぞれのトピックを識別するには，１文字か２文字を入力すればそれで十分である．

［Ｐ］オプションの指定されたＨＥＬＰを入力すると，23行ごとに表示が停止する動作は行なわれなくなる．

例：

```
A>HELP
```

上記のコマンドは，ＨＥＬＰが可能なすべてのトピックのリストを表示する．

```
A>HELP STAT
```

このコマンドはＳＴＡＴコマンドの一般的な情報を表示する．またサブトピックもすべて表示する．

```
A>HELP STAT OPTIONS
```

上記のコマンドにはサブトピック〝options〞が含まれている．それに対する応答として，ＨＥＬＰはＳＴＡＴコマンドのオプションに関する情報を表示する．

```
A>HELP ED
```

上記のコマンドはＥＤユーティリティに関する一般的な情報を表示する．

```
A>HELP ED COMMANDS
```

この形式のＨＥＬＰを入力すると，ＥＤの内部コマンドに関する情報が表示される．

4.10.1 HELP. HLPファイルへユーザのテキストの追加

CP/M-86にはHELP．CMDとHELP．HLPの2つの関連したファイルがある．HELP．CMDは HELP.HLP のテキストを処理し，スクリーン上に表示する．HELP．HLPは，ユーザ独自の情報を追加することのできるテキスト・ファイルであるが，直接HELP．HLPを編集することはできない．ユーザ・テキストを追加する前にHELP. CMD を使って，HELP．HLPをHELP．DATという名前のファイルに変換しなければならない．

HELP．HLPからHELP．DATへ，あるいはHELP．DATからHELP．HLPへの変換はHELPコマンドの下記の型式で行なう．

HELP 〔E〕
HELP 〔C〕

HELP 〔E〕コマンドは，省略時ドライブ上のHELP．HLPと呼ばれるファイルをアクセスし，そのヘッダー・レコードを削除し，新らしく省略時ドライブ上にHELP．DATと呼ばれるファイルを作成する．

HELP〔C〕コマンドは，省略時ドライブ上でユーザが編集したHELP．DATのファイルをアクセスし，新しいヘッダー・レコードを作成し，省略時ドライブ上にHELP．HLPのファイルを再構築する．この後HELP．CMDで新しく作成されたHELP．HLPファイルを表示することができる．

ユーザは定められた型式でHELP．DATのファイルに，トピックとサブトピックを追加しなければならない．HELP．DATファイル内のトピックの見出しの一般型式は下記の通りです．

\\\nTOPICNAME <cr>

3本のバック・スラッシュ(\\\) はトピックの識別子で，カラム1から始めなければならない．上記の型式内のnは1から9までの数値で，トピックのレベルを示す．主トピックは常にレベル1であり，最初のサブトピックはレベル2である．またその次のサブトピックのレベルは3で以下同様に最大レベル番号は9である．TOPICNAMEはユーザのトピックの名前で最大12文字まで使用できる．行の最後はキャリッジ・リターンで終る．

下記のガイドラインに従ってHELP．DATファイルを編集し，またそれにテキストの追加を行なう．

- トピックは英字の昇順に並んでなければならない．
- サブトピックはそれぞれの範囲内で英文字の昇順に並んでなければならない．
- レベルは1から9の範囲で示さなければならない．

HELP．HLPファイル内のトピックおよびサブトピックのいくつかの例を下記に示す．

\\\1NEW UTILITY <cr>
\\\2COMMANDS <cr>
\\\3EXAMPLES <cr>

上記の最初の例は，主トピックの型式を表わしている．2番目の例は主トピックに続く最初のサブトピックのレベル番号付けの方法を示している．3番目の例は，サブトピックの次のレベル番号付けの方法を示している．レベル番号2を持つトピック名はいずれも，その主トピックに対するサブトピックである．

HELP．CMDファイルを実行する時，トピック名が一意になるに充分な文字を入力しなければならない．サブトピックを参照する時は，サブトピック名と同時にトピック名も入力しなければならない．そうしなければ，どの主トピックに従属しているサブトピックか知ることができない．CP/M-86のプロンプトHELP>に続いてトピックおよびサブトピックを入力することができる．

HELP>ED COMMANDS

上記のHELP型式では，編集プログラムであるEDの情報を表示する．

4.11 PIP（周辺装置交換プログラム＝コピーファイル）コマンド

構文：

```
PIP dest-file{[Gn]}|dev=src-file{[options]}|dev{[options]}
```

タイプ：

トランジェント・ユーティリティ

目的：

PIPユーティリティは，あるディスクから別のディスク，またはあるユーザ番号から別のユーザ番号へ，1つまたは複数のファイルをコピーする．コピーの後，PIPはそのファイルの名前を変更することができる．またPIPは，複数のファイルを1つのファイルにまとめることができる．さらに文字ファイルをディスクからプリンタあるいは他の補助論理出力装置にコピーすることもできる．

PIPはコンソールまたは他の論理入力装置からの入力によって，ディスクにファイルを作成することができる．PIPは論理入力装置から論理出力装置へデータを転送することができる．したがって周辺装置変換プログラムという名前が付けられている．

4.11.1 単一ファイルのコピー

構文：

```
PIP d:{[Gn]} = source-filespec{[options]}

PIP dest-filespec{[Gn]} = d:{[options]}

PIP dest-filespec{[Gn]} = source-filespec{[options]}
```

目的：

この最初の形式は，ファイルをコピーするための最も簡単な方法を示すものである．

PIPは省略時ドライブまたは指定されたドライブにセットされているディスクのソース・ファイル記述によって，指定されているファイルを見つける．PIPはそのファイルをd：によって指定されているドライブにコピーし，ソース・ファイル記述と同じ名前をそのファイルに与える．必要に応じて［Gn］オプションを使用することにより，目的ファイル（dest-filespec）をnによって指定されるユーザ番号にセットすることができる．ソース・ファイル記述（source-filespe ）に対しては，複数のオプションを指定することができる．PIPオプションに関してはその項を参照されたい．

2番目の形式は最初の形式を変化させたものである．PIPはd：によって指定されるドライブにセットされているディスクからdest-filespecによって指定されるファイルを見つけ，それを省略時ドライブまたは指定されているドライブにコピーしdest-filespecと同じ名前を与える．

3番目の形式はコピーを行なった後，ファイルの名前を変更する方法を示したものである．この場合，ファイルを同じドライブおよび同じユーザ番号にコピーすることもできるし，異なるドライブあるいは異なるユーザ番号にコピーすることもできる．オプションに対する規則は同じである．PIPはsource-filespecによって指定されているファイルを見つけ，それをdest-filespecによって指定されている位置にコピーし，dest-filespecによって指定されている名前をそのファイルに割当て

PIPは特に［Gn］オプションが指定されていない限り，現在のユーザ番号を使ってコピーを行なう．

備考：

ＰＩＰを実行する場合には，前もってコピーしたいファイル（１つまたは複数）全体を保持できるだけの使用可能な空間が目的ディスクに存在するかどうかを確認しておかなければならない．目的ディスクの古いコピーを新しいコピーと置換える場合には，古いコピーを削除する前に，新しいコピーを記憶するための十分な領域が必要である（ＳＴＡＴユーティリティを参照）．

データ・ファイル全体がディスクの使用可能な空間に格納できるかどうかを確かめるために，データはまず一時ファイルにコピーされる．ＰＩＰは一時ファイルに対し目的ファイルのファイル名を割当て，さらにファイル型として＄＄＄を割当てる．コピー・オペレーションが成功すると，ＰＩＰは一時ファイルのファイル型＄＄＄を目的ファイルに指定されているファイル型に変更する．

コピー・オペレーションが成功し，目的ファイルと同じ名前を持つファイルが存在する場合には，同じ名前の古いファイルは一時ファイルの名前を変更する前に消去される．

ファイル属性（ＳＹＳ，ＤＩＲ，ＲＷ，ＲＯ）はファイルと一緒にコピーされる．

既存の目的ファイルが読取り専用（ＲＯ）にセットされている場合には，ＰＩＰはそれを削除してよいのかどうか質問するこれに対してユーザはＹまたはＮを応答する．読取り専用ファイルの上に重ねて書込む場合には，Ｗオプションを使用する．

各ソース名の後にＰＩＰオプションを指定することができる（下記のＰＩＰオプションを参照）．目的ファイル記述に対しては１つのオプションだけが有効である（[Ｇn]－ユーザ番号をｎにセットする）．オプションは角括弧で囲まれる．

ソース・ファイルに対しては複数のオプションを指定することができる．これらは一緒にまとめることもできるし，スペースで区切ることもできる．オプションを使用すれば，ファイルが正しくコピーされたかどうかを調べることができるし，システム（ＳＹＳ）属性を持つファイルを読取ったり，読取り専用ファイルに他のデータを書込んだり，ファイルを指定されているユーザ番号にセットしたり，小文字から大文字へ変換したり，いろいろな処理を行なうことができる．

例：

```
A>PIP B:=A:oldfile.dat

A>PIP B:oldfile.dat = A:
```

このコマンドのどちらの形式を入力しても，ＰＩＰはドライブＡからファイル oldfile. dat を読取り，その正しいコピーをドライブＢに書込む．このコマンドではソースまたは目的のドライブだけを指定しており，ファイル名はまったく指定されていないので，ＰＩＰの短い形式と呼ばれる．この形式を使用する場合には，ファイルをあるドライブあるいはあるユーザ番号から同じドライブおよび同じユーザ番号にコピーすることはできない．つまり目的ファイルを異なるドライブまたは異なるユーザ番号に書込まなければならない．詳しいことについてはＰＩＰオプションおよびＵＳＥＲコマンドの項を参照されたい．

２番目の形式を入力しても，最初のものとまったく同じ結果が得られる．ＰＩＰは単にソースとして指定されているドライブＡからファイル oldfile. dat を見つける．

```
A>PIP B:newfile.dat=A:oldfile.dat
```

このコマンドはファイル oldfile. dat をドライブＡからドライブＢにコピーし，その名前をnewfile. datに変更する．オリジナル・ファイルは oldfile. dat としてドライブＡにそのまま残される．この形式はコマンド行の両側にファイル名が指定されているので，ＰＩＰコマンドの長い形式である．

```
A>PIP newfile.dat = oldfile.dat
```

ＰＩＰのこの長い形式を使用すれば，あるドライブおよびユーザ番号（ＣＰ／Ｍ-86は自動的にユーザ０ー省略時ユーザ番号から開始するので通常ユーザ番号は０である）から，同じドライブおよび同じユーザ番号ファイルをコピーすることができる．このようにすれば同じファイルの２つのコピーを１つのドライブおよびユーザ番号に作成することができ，さらにそれぞれ異なる名前を付けることができる．

```
A>PIP B:PROGRAM.BAK = A:PROGRAM.DAT[G1]
```

上記のコマンドはファイルＰＲＯＧＲＡＭ．ＤＡＴをドライブＡのユーザ位置から，ドライブＢの現在選択されているユーザ番号にコピーし，ドライブＢのファイル型をＢＡＫに変更する．

```
B>PIP program2.dat = A:programl.dat[E V G3]
```

このＰＩＰコマンドは，ドライブＡの programl. dat という名前のファイルをコピーし，それをコンソールにエコー［Ｅ］し，２つのコピーがまったく同じであるかどうか調べ［Ｖ]，ドライブＡのユーザ３から programl. dat を入手［Ｇ３］する．目的ドライブは指定されていないので，ＰＩＰは自動的に省略時ユーザ番号およびドライブ，この場合にはドライブＢにファイルをコピーする．

4.11.2 複数ファイルのコピー

構文：

```
PIP d:{[Gn]} = {d:}wildcard-filespec{ [options]}
```

目的：

ソース指定の中でワイルド・カードを使用すると，ＰＩＰはファイルを１つずつ目的ドライブにコピーし，各ファイルのもとの名前をそのまま付ける．ＰＩＰはコピー・オペレーションが行なわれている間，各ファイル名の後に〝ＣＯＰＹＩＮＧ〞というメッセージを表示する．目的ドライブおよびユーザ番号がソースに指定されているものと同じである場合には，エラー・メッセージが出力され，コピー・オペレーションは中断する．

例：

```
A>PIP B:=A:*.CMD
```

このコマンドを入力すると，ＰＩＰはドライブＡにあるファイル型，ＣＭＤのファイルを，すべてドライブＢにコピーする．

```
A>PIP B:=A:*.*
```

このコマンドを入力すると，ＰＩＰはドライブＡのすべてのファイルをドライブＢにコピーする．このコマンドを使用すれば，配布されるディスクのバックアップ・コピーを作成することができる．ただしシステム・トラックからＣＰ／Ｍ-86システムをコピーすることはできない．システムをコピーするには，ＣＯＰＹＤＩＳＫを使用する．

```
A>PIP B:=A:PROG????.*
```

このコマンドを入力すると，ファイル型とは無関係に，ファイル名がＰＲＯＧから始まるすべてのファイルをドライブＡからドライブＢにコピーする．

```
A>PIP B:[G1]=A:*.A86
```

このコマンドを入力すると，ＰＩＰは省略時ユーザ番号（user コマンドを使ってユーザ番号を変更していない限り，ユーザＺＥＲＯ）でドライブＡに格納されているファイル型がA86であるすべてのファイルを，ユーザ番号１でドライブＢにコピーする．（ＤＩＲ，ＴＹＰＥ，ＥＲＡおよびその他のコマンドは，呼び出されたユーザ番号と同じユーザ番号のファイルだけをアクセスする．詳細はＵＳＥＲユーティリティを参照されたい)．

4.11.3 ファイルの組合せ

構文：

```
PIP dest-file{[Gn]}=src-file{[opt]},file{[opt]}{,file{[opt]}...}
```

目的：

PIPコマンドのこの形式を使用すれば，ソースに複数のファイルを指定することができる．PIPはソースに指定されているファイルを左から右へ順にコピーし，1つのファイルにまとめ，目的ファイル記述に示されている名前を付ける．この手順をファイル連結と呼ぶ．

目的ファイルの後に［Gn］オプションを使用すれば，そのファイルをnによって指定されているユーザ番号にセットすることができる．各ソース・ファイルに対して，1つまたは複数のオプションを指定することができる．

備考

ほとんどのオプションでは，PIPはファイルを1文字ずつコピーする．この場合PIPはファイルの終りを判断するためにCTRL-Zという文字を捜す．Gn，K，O，R，V，およびWを除き，どのPIPオプションも1文字ずつの転送が行なわれる．

論理装置との間でデータをコピーする場合にも，1文字ずつの転送が行なわれる．

文字転送を行なっているときにキーボードから任意のキーを入力すると，ファイル連結オペレーションは終了する．

ファイルを連結する場合，PIPは単にファイルの最後のレコードからファイルの最後の文字であるCTRL-Zを見つける．しかし，PIPが文字転送を行なっている時にCTRL-Zという文字が検出されると，そのオペレーションは停止する．

機械コード・ファイルを連結する場合［0］オプションを使用する．［0］オプションを指定すると，PIPは通常，ファイルの文字を示すCTRL-Zという文字がコードの中に組込まれても，その文字を無視する．

例：

```
A>PIP NEWFILE=FILE1,FILE2,FILE3
```

3つのファイルFILE1，FILE2，FILE3が左から右へ結合され，NEWFILE．$$$にコピーされる．NEWFILE．$$$の名前はコピー・オペレーションが終了したときにNEWFILEに変更される．

ソース・ファイルと目的ファイルはすべて省略時ドライブAのディスクに格納される．

```
A>PIP B:X.A86 = Y.A86, B:Z.A86
```

ドライブAのY．A86がドライブBのZ．A86と組合わされ，ドライブBの一時ファイルX．$$$に書込まれる．ファイルX．SSSはPIPの実行が完了したときにX．A86に変更される．

4.11.4 補助装置との間でのファイル・コピー

構文：

```
PIP dest-filespec {[Gn]} = source-filespec {[options]}
    AXO:                   AXI: {[options]}
    CON:                   CON: {[options]}
    PRN:                   NUL:
    LST:                   EOF:
```

目的：

この形式はPIPコマンド行の特殊な形式であり，ファイルをディスクから装置へ，あるいは装置からディスクへ，あるいはある装置から別の装置へコピーする場合に使用する．ファイルにはプリント可能な文字が含まれていなければならない．各周辺装置にはデータを転送できるソース装置，またはデータを受取ることのできる目的装置を示す“論理”名を割当てることができる．ファイル名と混同されないように，各論理装置名の後には（：）を指定する．任意のキーを押すと，ソースまたは目的論理装置を使用しているコピー・オペレーションは中断される．

論理装置名は次の通りである．

ＣＯＮ：　コンソール：に割当てられる物理装置．ソースとして使用される場合には，一般にキーボードである．宛先として使用される場合には，一般にスクリーンである．

ＡＸＩ：　補助入力または出力装置

ＡＸＯ：　補助出力装置

ＬＳＴ：　ＬＳＴ：に割当てられる目的装置．一般にプリンタである．

次の3つの装置名は特殊な意味を持つ．

ＮＵＬ：　40個の16進数０を作成するソース装置．

ＥＯＦ：　1つのＣＴＲＬ-Ｚを作成するソース装置（ＣＰ／Ｍ-86のファイル終了マーク）

ＰＲＮ：　各8桁ごとにタブ拡張を持ち，さらに行番号を印刷し，60行ごとに改ページを行なうプリンタ装置．

例：

```
B>PIP PRN:=CON: MYDATA.DAT
```

コンソール入力装置．一般にはキーボードから文字が読取られ，それが直接，プリンタ装置に出力される．キーボード入力が終了したことをPIPに示すには，ＣＴＲＬ-Ｚという文字を入力する．この文字が入力されると，ＰＩＰはさらにドライブBのファイルMYＤＡＴＡ．ＤＡＴから文字データを読取るＰＲＮ：が目的装置なので，タブが拡張され，行番号が追加され，60行ごとに改ページが行なわれる．

```
A>PIP B:FUNFILE.SUE = CON:
```

ＣＴＲ：がＣＯＮ：に割当てられている場合には，コンソールからタイプした文字はすべてドライブBのファイルＦＵＮＦＩＬＥ．ＳＵＥに書込まれる．ＣＴＲＬ-Ｚをタイプするとキーボードからの入力は終了する．

```
A>PIP LST:=CON:
```

ＣＲＴ：がＣＯＮ：に割当てられている場合には，キーボードから入力された文字がリスト装置，一般にはプリンタに書込まれる．ＣＴＲＬ-Ｚをタイプするとキーボードからの入力は終了する．

```
A>PIP LST:=B:DRAFT.TXT[T8]
```

ドライブＢのファイルＤＲＡＦＴ．ＴＸＴがプリンタ装置に書込まれる．タブ文字は８の倍数である最も近いカラムに拡張される．

```
A>PIP PRN:=B:DRAFT.TXT
```

このコマンドを入力すると，ＰＩＰはファイルＤＲＡＦＴ．ＴＸＴをリスト装置に書込む．自動的にタブが拡張され，行番号が追加され，60行ごとに改ページが行なわれる．

4.11.5 多重コマンド・モード

構文：

ＰＩＰ

目的：

ＰＩＰコマンドのこの形式は，ＰＩＰユーティリティを開始しＰＩＰがユーザ・メモリに存在する間，ユーザは複数のコマンド行をタイプすることができる．

備考：

コマンド行の入力を受取れる状態になると，ＰＩＰは（＊）をスクリーンに表示する．

このプロンプトが表示されたら，これまでにＰＩＰのいろいろな形式で説明した正しいコマンド行をタイプすることができる．

アスタリスク・プロンプトの後にＲＥＴＵＲＮキーを入力すると，ＰＩＰは終了する．ＲＥＴＵＲＮキーだけを入力すると，ＰＩＰはオペレーションを終了しＣＰ／Ｍ-86システム・プロンプトが出力される．

例：

```
A>PIP
*NEWFILE=FILE1,FILE2,FILE3
*APROG.CMD=BPROG.CMD
*A:=B:X.A86
*B:=*.*
*
```

このコマンドはＰＩＰプログラムをロードする．ＰＩＰコマンド入力プロンプト（＊）はＰＩＰがコマンドを受取れる状態であることをユーザに示す．コマンドをこのように入力して得られる結果は，前の例に示されている結果と同じである．この場合，コマンド・テイルにコマンド行が含まれており，各コマンドごとにＰＩＰがメモリにロードされるわけではない．

4.11.6　PIPのオプションの使い方

目的：

オプションを使用すれば，ソース・ファイルを特殊な方法で処理することができる．タブ文字を拡張したり，大文字から小文字に変換したり，テキストの一部だけを抽出したり，コピーが正しいかどうかを調べたり，さらにいろいろな操作を行なうことができる．

PIPオプションは下記に示す通りである．数値を示すために〝n〟が使用されており，CTRL-Zによって終了する一連の文字を示すために〝S〟が使用されている．オプションは，それが影響を及ぼすファイルまたは装置のすぐ後に指定しなければならず，また角括弧［ ］で囲まなければならない．数値を必要とするオプションの場合，文字と値の間にブランクを指定してはならない．

目的ファイル記述の後には，［Gn］オプションを指定することができる．ソース・ファイルまたはソース装置の後には，オプション・リストを指定することができる．オプション・リストとはオプションを示す文字と数値が並べられているものであり，ブランクによって区切られ，角括弧［ ］で囲まれている．

表4-3．PIPオプション

オプション	機能
Dn	カラムnから後の文字を削除する このパラメータはソース・ファイルに目的装置で取扱うことのできない長い行が含まれている場合に，ソース・ファイルの後に指定する．例えば目的装置80文字のプリンタや，1行の表示幅の狭いコンソールを使用する場合である．数値nは目的装置の最大カラム数である．
E	コンソールに文字をエコーする． このパラメータがソース名の後に指定されている場合には，PIPはコピーと同時にソース・データをコンソールに表示する．ソースには文字データが含まれていなければならない．
F	フォーム・フィード（改ページ）が行なわれないようにする． このパラメータがソース名の後に指定されている場合には，PIPはソース・データの中に組込まれているすべてのフォーム・フィードを除去する．ソース・ファイルで1ページの長さとしてセットされているフォーム・フィードを目的ファイルで他のページ・サイズに変更したい場合には，Fコマンドを使ってソース・ファイルのフォーム・フィードを削除し，Pコマンドを使って目的ファイルに新しいフォーム・フィードを追加する．
Gn	ユーザ番号をnにセットする． このパラメータがソース名の後に指定されている場合には，PIPはユーザ番号nのディレクトリからソース・ファイルを見つける．このオプションが目的名の後に指定されている場合には，PIPは目的ファイルをユーザ番号nにセットする．番号は0から15の範囲でなければならない．
H	16進データの転送． PIPはすべてのデータが正しいIntel 16進ファイル形式であるかどうかを調べる．エラーが検出されるとコンソールにエラー・メッセージが出力される．
I	Intel 16進形式ファイルを転送しているときに，：00レコードを無視する． Iオプションは自動的にHオプションをセットする．
L	ソース・ファイルの大文字を小文字に変換して目的ファイルにコピーーする． このパラメータはソース装置またはソース・ファイル名の後に指定する．

N　行番号を目的ファイルに追加する．

このパラメータがソース・ファイル名の後に指定されている場合には，PIPはコピーされる各行に行番号を追加する．この場合，行番号は1から始まり，1ずつ増加する．行番号の後にはコロンが置かれる．N2が指定されている場合，PIPは行番号に先行ゼロを追加し，番号の後にタブを挿入する．Tパラメータもセットされている場合には，PIPは次のタブに移動する．

O　機械コード（文字以外のデータつまり印刷できない文字）ファイルのためのオブジェクト・ファイルの転送．

PIPは連結および転送を行なっている間，ファイルの終りを示すCTRL-Zをすべて無視する．オブジェクト・コード・ファイルを結合する場合には，このオプションを使用する．

Pn　ページの長さをセットする．

nは1ページの行数を示す．このパラメータがソース・ファイルを変更する場合には，PIPは目的ファイルの始めと各n行毎にページ・エジェクト（改ページ）を追加する．n＝1の場合またはnが指定されていない場合には，PIPは60行ごとのページ・エジェクトを挿入する．Fオプションも指定されている場合には，PIPはソース・データのフォーム・フィード（改ページ）を無視し，nによって指定されているページの長さで目的データに新しいフォーム・フィードを挿入する．

Qs　ストリングSをコピーした後，ソース装置からのコピーを中止する．

Sパラメータと一緒に使用すると，このパラメータはソース・ファイルの一部分だけを抽出することができる．ストリング引数はCTRL-Zによって終了しなければならない．

R　システム（SYS）ファイルを読取る．

一般に，PIPはディスク・ディレクトリにシステム属性のあるファイルを無視する．しかしこのパラメータがソース・ファイル名の後に指定されている場合には，PIPはその属性も含めてシステム・ファイルを目的ファイルにコピーする．

Ss　ストリングSの位置からソース装置からのコピーを開始する．

ストリング引数は，CTRL-Zによって終了しなければならない．Qパラメータと一緒に使用すると，このパラメータはソース・ファイルの一部だけを抽出することができる．オペレーションを開始するストリング，およびオペレーションを中止するストリングのどちらも目的ファイルに含まれる．

Tn　タブを拡張する．

このパラメータがソース・ファイル名の後に指定されていると，PIPは目的ファイルにタブ（CTRL-I）文字を挿入する．PIPはそれぞれのCTRL-Iを十分なスペースと置き換え，nによって割りきるカラムに次の文字を配置する．

U　ソース・ファイルの小文字を大文字に変換し，目的ファイルにコピーする．

このパラメータはソース装置またはソース・ファイル名の後に指定する．

V　データが正しくコピーされたかどうかを調べる．

PIPは目的とソースを比較し，データが正しく書込まれたかどうかを確認する．目的はディスク・ファイルでなければならない．

W　RO（読取り専用）属性を持つファイルにコピーする．

通常PIPコマンド・テイルに目的として既存のROファイルが含まれている場合には，PIPはそのファイルの上からデータを書込んでよいかどうか確認するための質問をコンソールに表示する．このパラメータがソース名の後に指定されている場合には，PIPはROファイルをコピーし，コンソールに質問は表示されない．コマンド・テイルに複数のソース・ファイルが含まれている場合には，リスト中の最後のファイルにだけ，このパラメータを指定すればよい．

Z　　パリティ・ビットをゼロにする．
このパラメータがソース名の後に指定されている場合には，ＰＩＰは目的ファイルの各データ・バイトのパリティ・ビットをゼロにセットする．ソースには文字データが含まれていなければならない．

例：

```
A>PIP NEWPROG.A86=CODE.A86[L], DATA.A86[U]
```

このコマンドは，ドライブAの2つのファイルCODE．A86とDATA．A86を結合することにより，ドライブAにファイルNEWPROG．A86を作成する．コピー・オペレーションを行なっている間，CODE．A86は小文字に変換され，一方DATA．A86は大文字に変換される．

```
A>PIP CON:=WIDEFILE.A86[D80]
```

このコマンドは，文字ファイルWIDEFILE．A86をドライブAからコンソール装置に書込むが，80桁より後の文字はすべて削除される．

```
A>PIP B:=LETTER.TXT[E]
```

ドライブAのファイルLETTER．TXTは，ドライブBのLETTER．TXTにコピーされる．LETTER．TXTというファイルは，コピー・オペレーションが行なわれるときにスクリーンにも表示される．

```
A>PIP LST:=B:LONGPAGE.TXT[FP65]
```

このコマンドは，ファイルLONGPAGE．TXTをドライブBからプリンタ装置に書込む．ファイルが書込まれるときにフォーム・フォード（改ページ）文字が削除され，目的ファイルの始めと65行おきに再挿入される．

```
B>PIP LST:=PROGRAM.A86[NT8U]
```

このコマンドは，ファイルPROGRAM．A86をドライブBからプリンタ装置に書込む．Nというパラメータは，ＰＩＰに対して各行に行番号を付けるように指示するものである．Ｔ８というパラメータは８カラムおきにタブを拡張する．Ｕパラメータはファイルが印刷されるときに，小文字を大文字に変換するパラメータである．

```
A>PIP PORTION.TXT=LETTER.TXT[SDear Sir^Z QSincerely^Z]
```

このコマンドは，コピー・オペレーションを開始する前に，一連の文字〝Dear Sir〟を捜すことにより，ドライブＡからLETTER．TXTというファイルの一部だけを抽出する．そのような文字ストリングが見つかると，ソース・ファイルから〝Sincerely〟という文字ストリングが見つかるまで，すべての文字がドライブAのPORTION．TXTにコピーされる．

```
B>PIP B:=A:*.CMD[VWR]
```

このコマンドは，ファイル型がCMDであるすべてのファイルを，ドライブAからドライブBにコピーする．Ｖというパラメータは，データが正しくコピーされたかどうかを確認するために，目的ファイルを読取るように指示するものである．Wというパラメータは，RO（読取り専用）というマークの付いている目的ファイルにデータをコピーするように指示するものである．Rというパラメータは，SYS（システム）属性を持つファイルをドライブAから読取るように指示する．

4.12　REN（名前の変更）組込みコマンド

構文：

```
REN {d:}newname{.typ} = oldname{.typ}
```

タイプ：

組込みコマンド

目的：

REN組込みコマンドを使用すれば，ディスクのディレクトリにカタログされているファイルの名前を変更することができる．

ファイル名oldnameはディスクに既に存在するファイルを示す．ファイル名newnameはディスクのディレクトリに存在しないファイルである．RENコマンドは，oldnameによって指定されたファイルの名前をnewnaweによって指定された名前に変更する．

備考：

RENではファイルのコピーが作成されるわけではない．RENは単にファイルの名前を変更するだけである．

ドライブ記述子を省略した場合には，RENは名前が変更されるファイルが省略時ドライブに存在するものとみなす．

newnameの一部としてドライブ記述子を指定することができる．どちらのファイル記述も，ドライブを指定している場合には，同じドライブでなければならない．

oldnameによって指定されるファイルが存在しない場合には，RENはスクリーンに次のメッセージを出力する．

```
NO FILE
```

newnameによって指定されるファイルがディレクトリに既に存在する場合には，RENはスクリーンに次のメッセージを表示する．

```
FILE EXISTS
```

例：

A>REN NEWASM.A86=OLDFILE.A86

ドライブAのファイルOLDFILE. A86がNEWASN. A86に変更される．

B>REN A:X.PAS = Y.PLI

ドライブAのファイルY. PLIが，X. PASに変更される．

A>REN B:NEWLIST=B:OLDLIST

ドライブBのファイルOLDLITSTがNEWLISTに変更される．2番目のドライブ記述子B：は最初のドライブ記述子から判断することができるので，この例では必ずしも指定する必要はない．したがって上記のコマンド行の結果は同じである．

A>REN B:NEWLIST=OLDLIST

4.13 STAT（ステータス）コマンド

構文

```
STAT
STAT  d:=RO
STAT filespec {RO|RW|SYS|DIR|SIZE}
STAT {d:}DSK: | USR:
STAT VAL: | DEV:
```

タイプ：

トランジェント・ユーティリティ

目的：

いろいろな形式のＳＴＡＴユーティリティ・コマンドは，コンピュータに関連するディスク・ドライブ，ファイルおよび装置に関する情報をユーザに提供する．ＳＴＡＴを使用すれば，ファイルおよびドライブの属性を変更することができる．またＳＴＡＴ論理装置名に物理装置を割当てることができる．

ファイル記述（filespec)の後に続くオプションは角括弧［ ］で囲むこともできるし，その前にドル記号＄を付けることもできる．また，上記に示されているように，区切り文字をまったく付けなくてもかまわない．

備考：

″ＲＷ″という表記は、ドライブが読み書き状態なので，データをディスクから読取ったりディスクに書込んだりすることができることを示す．

″ＲＯ″という表記は，ドライブが読取り専用状態なので，データをディスクから読取ることはできるが，ディスクに書込むことはできないことを示す．

ドライブは特に指定されない限り，省略値により読み書き状態であるが，ドライブをＲＯにセットしたり，ディスクを交換するときにＣＴＲＬ-Ｃをタイプし忘れたりすると読取り専用状態になる．

4.13.1 ドライブでの読取り専用ステータスのセット

構文：

```
STAT d:= RO
```

目的：

この形式のＳＴＡＴコマンドを使用すれば，ドライブを読取り専用ステータスにセットすることができる．ドライブを読み書きステータスに戻すにはCTRL-Cを使う．

例：

```
A>STAT B:= RO
```

このコマンド行は，ドライブＢを読取り専用ステータスにセットする．

4.13.2　ディスク未使用領域の表示

構文：

```
STAT {d:}
```

目的：

コマンド・テイルの指定されていないＳＴＡＴは，すべてのオンライン・ディスク上で使用可能な未使用格納領域の量を報告する．この形式のＳＴＡＴコマンドは，ＣＰ／Ｍ-86が最後に起動または再ロードされた後でアクセスされたディスクの使用可能な領域を報告する．特定のディスクの使用可能な領域の大きさを知るには，コマンド・テイルにドライブ記述子を指定しなければならない．

備考：

ドライブ記述子にオンラインではないドライブが指定されている場合には，ＣＰ／Ｍ-86はこのドライブをオンライン・ステータスにセットする．

この形式のＳＴＡＴコマンドは，下記の形式でスクリーンに情報を表示する．

```
d: RW, Free Space: nnK
```

ｄはドライブ記述子であり，ｎはｄによって指定されるドライブにセットされているディスクの使用可能な領域の大きさを，キロバイトで表わしたものである．

例：

```
A>STAT
```

２つのディスク・ドライブに，それぞれアクティブなディスクがセットされているとしよう．またドライブＡには使用可能な領域が16ｋ（16,384）バイトあり，ドライブＢには32ｋ（32,728）バイト存在するとしよう．ドライブＡはＲＷで，ドライブＢはＲＯにセットされている．この場合ＳＴＡＴコマンドを実行すると下記のメッセージがスクリーンに表示される．

```
A: RW, Free Space: 16K
B: RO, Free Space: 32K
```

```
A>STAT B:
```

ドライブＢが読取り専用ステータスにセットされており，プログラムおよびデータを格納できる使用可能な格納領域が98キロバイト存在するとしよう．すると下記のメッセージがスクリーンに表示される．

```
B: RO, Free Space: 98K
```

4.13.3 ファイル一使用領域とアクセス・モードの表示

構文：

```
STAT filespec {SIZE}
```

目的：

この形式のＳＴＡＴコマンドは，指定されたファイルが使用している領域の大きさをキロバイト単位で表示する．またファイルのアクセス・モードも表示する．ＳＴＡＴでは，コマンド・テイルのファイル名とファイル型の部分にワイルド・カードを指定することができる．ファイル記述にワイルド・カードを指定すると，ＳＴＡＴコマンドは省略時ドライブまたは指定されたドライブから，対応するファイルのリストをファイル特性と共にアルファベット順に表示する．

ファイル記述（filespec）の後のＳオプションは角括弧［ ］で囲むこともできるし，その前にドル記号＄を付けることもできる．また，上記に示されているように区切り文字をまったく使用しなくてもかまわない．

ＣＰ／Ｍ-86では，次の4つのファイル・アクセス・モードがサポートされている．

ＲＯ　ファイルは読取り専用属性を持っており，データをファイルから読取ることはできるが，ファイルの内容を変更することはできない．

ＲＷ　ファイルは読み書き属性を持っており，データをファイルに書込んだり，ファイルから読取ったりすることができる．

ＳＹＳ　ファイル〝システム〟属性を持っている．システム・ファイルは，ＤＩＲ（ディレクトリ）の表示では表示されない．システム（ＳＹＳ）ファイルを表示するにはＤＩＲＳを使用する．システム属性を持つファイルも含めてすべてのファイルを表示するには，ＳＴＡＴコマンドを使用する．ＳＴＡＴコマンドではシステム・ファイルはカッコで囲まれて表示される．

ＤＩＲ　ファイルは〝ディレクトリ〟属性を持ち，ＤＩＲ（ディレクトリ）表示で表示される．

ファイルはＲＯまたはＲＷ属性，それにＳＹＳまたはＤＩＲ属性を持つ．ＳＴＡＴコマンドを使って特に変更しない限り，省略値によりファイルはＲＷおよびＤＩＲ属性を持つ．

ＳＴＡＴコマンドのこの形式は，5つの見出しの下にファイル特性のリストを出力する．

- 最初のカラムには，ファイルが使用しているレコード数が表示される．この場合，各レコードの長さは128バイトである．この値は〝Recs.〟という見出しの下に表示される．
- 2番目のカラムには，ファイルが使用している領域の大きさがキロバイト（1024バイト）単位で表示される．この値は〝Bytes.〟の下に表示される．
- 3番目のカラムには，ファイルが使用しているディレクトリ項目の数が表示される．この値は〝FCBs〟というカラムの下に表示される．ＦＣＢ（ファイル・コントロール・ブロック）はディレクトリ項目の別の名前である．
- アクセス・モードは，〝Attributes〟カラムの下に表示される．
- ドライブ記述子，ファイル名およびファイル型から構成されるファイル記述は，スクリーンの〝Name〟の下に表示される．

備考：

ドライブ記述子が指定されているとき，そのドライブがアクティブでなければ，CP／M-86はそのドライブをアクティブ・ステータスにセットする．

STATに対し，各ファイルの〝仮想ファイル・サイズ〟を計算するように指示するには，sizeを使用する．順次ファイルの場合には仮想ファイル・サイズと実ファイル・サイズは等しいが，ランダム・モードで書込まれたファイルの場合には，両方のサイズは異なっている．SIZEを使用する場合には，さらに〝Size〟というカラムがスクリーンに表示される．このカラムに表示される値は，そのファイルに割振られているレコードの中で実際にデータが書込まれているレコード数と，書込まれていないレコード数を示す．

コマンドSTAT＊・＊を入力すると，STATはディレクトリを調べ，2つのファイルが同じディスク領域を共有していないかどうかを確認する．つまり指定されたファイルが，ディレクトリの中で別のファイルによって占有されているディスクの格納領域を共有していないかどうかを確認するのである．格納領域の割当てが重複していることが検出されると，STATは次のメッセージを表示する．

```
Bad Directory on d:
Space Allocation Conflict:
User nn  d:filename.typ
```

STATは，領域を重複して割当てられているファイルの名前とユーザ番号を印刷する．したがって，複数のファイルがリストされる．このような場合には表示されたファイルを消去し，それからCTRL-Cをタイプすることが望ましい．

コマンド・テイルにワイルド・カード文字が表示されている場合には，常にSTATは完全なディレクトリ検査を行なう．

例：

A>STAT MY*.*

このコマンドはファイル型にかかわらず，MYという文字から始まる名前を持つすべてのファイルの特性を表示するように，STATに指示する．下記の3つのファイルがファイル記述を満足するとしよう．するとスクリーンには次のメッセージが表示される．

```
Drive B:                                  User 0
Recs   Bytes   FCBs  Attributes           Name
 16     2K      1      Dir RW     B:MYPROG   .A86
  8     1K      1      Dir RO     B:MYTEST   .DAT
 32    18K      2      Sys RO     B:MYTRAN   .CMD

Total: 21K      4

B: RW, Free Space: 90K
```

A>STAT MY*.* SIZE

このコマンドは前のコマンドと同じ動作をするが，スクリーンに〝Size.〟というカラムも表示される．MYTEST. DATというファイルがレコード番号8から15までランダム・アクセスを使って書込まれており，最初の8つのレコードにはデータが書込まれていないとしよう．この場合，仮想ファイル・サイズは16レコードであるが，実際にデータが書込まれているレコードは8レコードである．スクリーンには次のメッセージが表示される．

```
Drive B:                                         User 0
Size   Recs   Bytes   FCBs   Attributes          Name
 16     16     2K      1        Dir  RW     B:MYPROG   .A86
 16      8     1K      1        Dir  RO     B:MYTEST   .DAT
 32     32    18K      2        Sys  RO     B:MYTRAN   .CMD

Total:        21K      4

B: RW, Free Space: 90K
```

4.13.4 ファイルアクセス・モード（属性）のセット

構文：

```
STAT filespec RO |RW |SYS |DIR
```

目的：

この形式のSTATコマンドを使用すると，1つまたは複数のファイルのアクセス・モードをセットすることができる．ファイル記述（filespec）の後に続くオプションは角括弧［ ］で囲むこともできるし，その前にドル記号＄を付けることもできる．また，上記に示されているように区切り文字をまったく使用しなくてもかまわない．

既に説明したように，次の4つのアクセス・モードを使用することができる．

```
RO
RW
SYS
DIR
```

備考：

ファイル記述の中に指定されているドライブがアクティブでない場合には，CP／M-86はまずそのドライブをオンライン状態にセットする．

ファイルはROまたはRWアクセス・モードのいずれかを持つことができるが，両方の属性を持つことはできない．同様にファイルはSYSまたはDIRアクセス・モードのいずれかを持つことができるが，両方を持つことはできない．

例：

```
A>STAT LETTER.TXT RO
```

このコマンドは省略時ドライブのファイルLETTER．TXTのアクセス・モードをROにセットする．このファイルが存在する場合には，スクリーンに下記のメッセージが表示される．

```
LETTER.TXT set to RO
```

次のコマンドは，

```
B>STAT A:*.COM SYS
```

ファイル型がCOMであるドライブAのすべてのファイルのアクセス・モードをSYSにセットする．3つのコマンド・ファイルPIP，ED，ASM-86がドライブAに存在するとすると，下記のメッセージがスクリーンに表示される．

```
PIP.COM set to SYS
ED.COM set to SYS
ASM86 set to SYS
```

4.13.5 ディスク・ステータスの表示

構文：

```
STAT {d:}DSK:
```

目的：

この形式のSTATコマンドは，すべてのオンライン・ディスクのディスク・システムに関する内部情報を表示する．ドライブが指定されている場合には，そのドライブはオンライン・ステータスにセットされる．

このコマンドが表示する情報は，高度なプログラミングを行なう場合には役に立つが，CP／M-86を日常使用する場合には必ずしも必要としない．

例：

```
A>STAT DSK:
```

このSTATコマンドは，下記の形式でドライブAに関する情報を表示する．nと示されている所には実際の数値が示される．

```
    A: Drive Characteristics
nnnn: 128 Byte Record Capacity
nnnn: Kilobyte Drive  Capacity
nnnn: 32 Byte Directory Entries
nnnn: Checked  Directory Entries
nnnn: 128 Byte Records/Directory Entry
nnnn: 128 Byte Records/Block
nnnn: 128 Byte Records/Track
nnnn: Reserved Tracks
```

```
A>STAT B:DSK:
```

このコマンドでは，前の例で示されている情報と同じ情報がドライブBに関して表示される．

4.13.6 アクティブ・ファイルを持つユーザ番号の表示

構文：

```
STAT {d:}USR:
```

目的：

この形式のSTATコマンドを使用すれば，指定されたドライブにセットされているディスクにファイルを持つユーザ番号を判断することができる．

ユーザ番号は，CP／M-86のもとで作成されたファイルに割当てられるものである．この形式のSTATコマンドを使用すれば，ディスク上のアクティブなユーザ番号を知ることができる．

例：

```
A>STAT USR:
```

このコマンドはドライブAにセットされているディスクのアクテブ・ファイルを持つユーザ番号を表示する．

4.13.7 STATコマンドと装置名の表示

構文：

```
STAT VAL:
```

目的：

STAT VAL：は，STATコマンドの一般形式を表示する．またCP／M-86の4つの論理装置名をそれぞれ割当てることのできる物理装置名も表示する．

例：

STAT VAL：は下記のメッセージを表示する．

```
A>STAT VAL:
STAT 2.1

Read Only Disk: d:=RO
Set Attribute: d:filename.typ [ro] [rw] [sys] [dir]
Disk Status  : DSK: d:DSK:
User Status  : USR: d:USR:
Iobyte Assign:
CON: = TTY: CRT: BAT: UC1:
AXI: = TTY: PTR: UR1: UR2:
AXO: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
A>
```

4.13.8 物理装置の表示と論理装置への割当て

構文：

```
STAT DEV:
STAT logical device: = physical device:
```

目的：

STAT DEV：は，CP／M-86の4つの論理装置名であるCON：，RDR：，PUN：，LST：の現在の割当てを表示する．これらの現在の割当てを変更するには，上記のSTATコマンドの2番目の形式を使用する．コマンドSTAT VAL：は，各論理装置名に割当てることのできる物理装置名を表示する．STAT VAL：の表示の中の〝Iobyte Assign〞というカラムを参照されたい．

物理装置を論理装置に割当てる場合，STATは論理装置名にIobyteと呼ばれる0から3までの値を割当てる．
表示された物理装置名はすべて適切な論理装置名に割当てることができる．しかし，コンピュータで正しい入出力ポートを使用し，正しいケーブルを使ってコンピュータと装置を接続し，さらにそれぞれの物理装置に対して正しいIO（入出力)ドライバを使用しない限り，この割当ては機能しない．

物理装置ドライバはBIOS（Basic I／O System）中にインプリメンテーションされなければならない．IObyteを読取り，解釈しなければならない．それぞれ該当するドライバは論理入出力ルーチンにジャンプしなければならない．外部物理装置の取扱いについてのさらに詳しい情報は，CP／M-86システムガイドを参照されたい．

例：

```
A>STAT CON: = CRT:
```

上記のコマンドは，物理装置名ＣＲＴ：を論理入力装置名ＣＯＮ：に割当てる．これは一般にコンソールを示す．

```
A>STAT LST: = LPT:
```

このコマンドは，物理装置名ＬＰＴ：を論理出力装置名ＬＳＴ：に割当てる．これは一般にリスト装置であるプリンタを示す．

4.14 SUBMIT（バッチ処理）コマンド

構文：

```
SUBMIT filespec { parameters... }
```

タイプ：

トランジェント・ユーティリティ

目的：

SUBMITユーティリティを使用すれば，CP／M-86で自動的に処理できるよう，コマンドをグループ化することができる．

通常，ユーザはコマンドを1度に1行ずつ入力する．しかし同じコマンド・シーケンスを何回も入力しなければならない場合には，SUBMITユーティリティを使って，コマンドを1つにまとめる方が容易である．コマンドを1つにまとめるためには，ファイルを作成し，このファイルの中にコマンドを書込んでおく．ファイルはファイル名によって識別され，またこのファイルのファイル型はSUBでなければならない．SUBMITコマンドを入力すると，SUBMITはファイル記述（filespec）で指定されたファイルを読取り，それをCP／M-86が解釈できるように準備する．

ファイル型がSUBであるファイルには，有効なCP／M-86のコマンドを書込むことができる．必要に応じてSUB ファイルにSUBMITパラメータを指定することができる．このパラメータはコマンド・テイルに含まれる値である．

SUBMITパラメータの形式は，ドル記号（＄）の後に1から9までの数値が続く．

```
$1
$2
$3
$4
$5
$6
$7
$8
$9
```

これらのパラメータは，ファイル型がSUBであるファイルのコマンド行の中のどこにでも挿入することができる．

SUBMITユーティリティはSUBMITファイル記述（filespec）の後にあるコマンド行を読取り，ファイル型SUBのファイルの中に含まれていたパラメータの代りに，コマンド・テイルにタイプした項目を使用する．この置換えが完了すると，SUBMITは各コマンドをユーザがタイプしているかのように，1行ずつファイルをCP／M-86に送る．

備考：

コマンド・テイルの各項目は英字，数字および特殊文字から構成される．項目は1つまたは複数のブランクで区切られる．

コマンド・テイルの中の最初のワードは＄1を置換え，2番目のワードは＄2というように最後のパラメータまで置換えが行なわれる．

SUBファイルのパラメータよりもコマンド・テイルに入力した項目の数の方が少ない場合，残りのパラメータはコマンド行から除去される．

SUBファイルのパラメータの数よりコマンド・テイルで入力した項目の数の方が多い場合には，コマンド・テイルの残りのワードは無視される．

ＳＵＢＭＩＴは＄＄＄．ＳＵＢという名前のファイルを作成する．このファイルには置換えによって作成されたコマンド行が記憶される．

ＳＵＢファイルの最後の行の読取りが終了すると，バッチ・コマンド処理は停止する．ＣＴＲＬ-ブレークを入力すると，ＳＵＢＭＩＴの処理は停止する．また，CP／M-86がコマンド入力プロンプトであるA>を出力した後，任意のキーを入力すれば，ＳＵＢファイルの終りに到達する前にバッチ処理は終了する。

ＣＰ／M-86がすべてのコマンド行を処理したら，ファイル＄＄＄．ＳＵＢは自動的に削除される．

ＳＵＢファイルの中にさらにＳＵＢＭＩＴコマンドを指定することはできない．しかし，SUBファイルの最後のコマンドとしてＳＵＢＭＩＴコマンドを指定し，別のＳＵＢファイルに〝連結〟することは可能である．

ファイル型ＳＵＢのファイルに実際のドル記号（＄）を含むには，２つのドル記号（＄＄）をタイプする．SUBMITユーティリティは，ＳＵＢファイルの中の＄パラメータをコマンド・テイル項目と置換えるときに１つの＄記号と置換える．

例：

```
A>SUBMIT SUBFILE
```

ファイルSUBFILE．ＳＵＢがドライブAのディスクに存在し，下記のコマンド行が格納されているとしよう．

```
DIR *.COM
ASM86 X $$SB
PIP LST:= X.PRN[T8D80]
```

上記のＳＵＢＭＩＴコマンドは，ＳＵＢＦＩＬＥ．ＳＵＢに含まれているコマンドを処理のために順にＣＰ／M-86に送る．ＣＰ／M-86はまずDIRコマンドを実行し，次にX．A86をアセンブルする．ＡＳＭ-86が完了すると，PIPコマンド行が実行される．

```
A>SUBMIT B:ASMCOM X  8 D80  SZ  <--these command tail
                                   items are assigned
                  $1 $2  $3  $4  <--to these SUB file $n
                                   parameters.
```

ＡＳＭＣＯＭ．ＳＵＢがドライブBに存在し，そのファイルに次のコマンドが含まれているとしよう．

```
ERA $1.BAK
ASM86 $1 $$$4
PIP LST:= $1.PRN[T$2 $3 $5]
```

ＳＵＢＭＩＴユーティリティはこのファイルを読取り，ＳＵＢファイルのパラメータの代りにコマンド・テイルの各項目を置換える．

```
ERA X.BAK
ASM86 X $SZ
PIP LST:= X.PRN[T8 D80]
```

ＣＰ／M-86は，これらのコマンドを上から下へ順に実行する．

4.15 TOD（日時の表示とセット）コマンド

構文：

TOD {time-specification | P }.

タイプ：

トランジェント・ユーティリティ

目的：

TODユーティリティを使用して，日時を調べたり，セットすることができる．

CP／M-86を起動すると，日付および時刻はBDOSの作成日にセットされる．TODを使用すれば，この初期値を現在の日付および実際の時刻に変更することができる．

日付は1から12までの月，1から31までの日（これは月によって異なる），および西暦の下2桁を示す2桁の年によって表現される．

時刻は24時間表現であり，午前は00から11までの値で，午後は12から23までの値で示される．

次のコマンドを使用すれば，

```
TOD
```

現在の日付および時刻を次の形式で知ることができる．

曜日　月／日／年　時：分：秒

例えば，TODコマンドに対する応答として画面に次のような情報が表示される．

```
12/05/81          09:15:37
```

次のコマンド形式を使用すれば，

```
TOD time-specification
```

日付および時刻をセットすることができる．この場合，時刻記述の形式は次の通りである．

月／日／年　時：分：秒

この形式のコマンド行の例を下記に示す．

```
TOD 02/09/81 10:30:00
```

時刻を正しくセットできるように，TODユーティリティは次のメッセージを出力する．

```
Press any key to set time
```

コマンド・テイルに指定した時刻になったら，ユーザは任意のキーを押す．するとその瞬間からTODは時刻のカウントを開始し，その応答として次の形式の情報を表示する．

```
02/09/81          10:30:00
```

次の形式のコマンドを使用すれば，

```
TOD P
```

日付および時刻が連続的にスクリーンに表示される．任意のキーを押せば連続表示は停止する．

備考：

ＴＯＤは，時刻記述が有効な日付および時刻を表現しているかどうかを確認する．

日付および時刻をセットしなくても，ＣＰ／M-86の正しいオペレーションには何の影響も与えない．

例：

```
A>TOD
```

このコマンドは，現在の日付および時刻をスクリーンに表示する．

```
A>TOD 12/31/81 23:59:59
```

このコマンドは，現在の日付および時刻を1981年の最後の秒にセットする．

4.16　ＴＹＰＥ（ファイルの表示）組込みコマンド

構文：

```
TYPE {d:}filename{.typ}
```

タイプ：

組込みコマンド

目的：

ＴＹＰＥ組込みコマンドは，文字ファイルの内容をスクリーンに表示する．

備考：

ファイル記述によって指定されているファイルがオンライン・ディスクに存在しない場合には，ＴＹＰＥは次のメッセージをスクリーンに表示する．

```
NO FILE
```

ファイルをスクリーンとプリンタの両方にリストするには，ＴＹＰＥコマンド行を入力する前にＣＴＲＬ-Ｐをタイプする．キーボード入力をプリンタに出力するのを停止する場合には，さらにＣＴＲＬ-Ｐをタイプする．

例：

```
A>TYPE MYPROG.A86
```

このコマンドは，ファイルMYPROG．A86の内容をスクリーンに表示する．

```
A>TYPE B:THISFILE
```

このコマンドは，ドライブBのファイルＴＨISFILＥの内容をスクリーンに表示する．

4.17 ＵＳＥＲ（ユーザ番号の表示とセット）組込みコマンド

構文：

```
USER { number }
```

タイプ：

組込みコマンド

目的：

ＵＳＥＲ組込みコマンドは，現在のユーザ番号を表示したり変更したりする．ディスク・ディレクトリは，〝ユーザ番号〟に応じて異なるグループに分けることができる．

備考：

ＣＰ／M-86を起動すると，その時のユーザ番号は0となる．このユーザ番号のもとで作成したファイルは，一般に他のユーザ番号のもとでアクセスすることができない．ただし，ＳＴＡＴコマンドを使って割当てられたシステム（ＳＹＳ）属性やＰＩＰコマンドを使用すれば，他のユーザ番号からでもアクセスすることができる．（PIPユーティリティのGパラメータを参照）．

次のコマンドを使用すれば，

```
USER
```

現在のユーザ番号を表示することができる．

次のコマンドを使用すれば，

```
USER number
```

現在のユーザ番号を変更することができる．ただし，Number は0から15の範囲の数値である．

アクティブなファイルを持つユーザ番号のリストを表示するには，次のコマンドを使用する．

```
STAT USR:
```

例：

```
A>USER
0
```

このコマンドは現在のユーザ番号を表示する．

```
A>USER 3
```

このコマンドは現在のユーザ番号を3に変更する．

第5章
ED，CP／M-86エディタ

5.1　EDの紹介

コンピュータを使って何かを行なおうとする場合には，コンピュータにデータを入力する方法つまり処理したいと思う情報をコンピュータに与える何らかの方法が必要である．このために一般的に使用されるプログラムのことを〝エディタ〟と呼ぶ．エディタはキーボードから入力されたデータをディスク・ファイルに格納する．CP／M-86のエディタの名前はEDである．EDを使用すれば，CP／M-86のテキスト・ファイルを容易に作成したり，変更したりすることができる．

CP／M-86のエディタを呼出すための正しいコマンド構文については，最初の節である〝EDの始動〟に示されている．EDを起動したら，編集したいテキストをディスク・ファイルからメモリに読取るコマンドを入力する．〝EDオペレーション〟の節では，このオペレーションに関する詳しい説明が示されており，容易にエディタに入ったり，エディタから抜け出したりすることのできる，基本的なテキスト転送コマンドについても説明されている．

〝基本的な編集コマンド〟の節には，ファイルを編集するコマンドの詳しい説明が示されている．〝EDコマンドの組合せ〟では，編集をさらに効率よく行なうための基本的なコマンドの組合せ方が説明されている．基本的なEDコマンドを使用すれば，どのようなファイルも編集できるが，EDには〝拡張EDコマンド〟の節に示されているように，さらに複雑な編集機能を実行するいくつかのコマンドがある．

編集セッションの間，EDは2種類のエラー・メッセージを出力することがある．〝EDエラー・メッセージ〟の項には，これらのメッセージのリストと一般的な編集エラー状況からどのように回復すればよいかを示す例が示されている．

5.2 EDの起動

構文:

```
ED filespec filespec
```

EDを起動するには，CP／M-86のプロンプトが出力された後，その名前を入力する．コマンドEDの後にファイル記述を指定しなければならない．そのファイル記述には，万能札文字を含んではならない．

```
A>ED MYFILE.TEX
```

上記の例のファイル記述MYFILE．TEXは，編集または生成されるファイルを指定する．ファイル記述の前にドライブ記述子を指定することもできるが，編集されるファイルが省略時ドライブに存在する場合には，ドライブ記述子を指定する必要はない．また，次の例に示すように，ファイル記述の後にドライブ記述子を指定することもできる．

```
A>ED MYFILE.TEX B:
```

このコマンドに対する応答として，EDは編集対象ファイルであるドライブAのMYFILE．TEXをオープンするが，編集された結果はドライブBのファイルへ格納される．

また，下記の例に示されているように編集した結果を異なるファイル名を持つファイルに書き込むこともできる．

```
A>ED MYFILE.TEX YOURFILE.TEX
```

異なるファイル名を持つファイルが既に存在していてはならない．そのような場合には，EDは次のメッセージを出力し処理を終了する．

```
Output File Exists, Erase It
```

EDがコマンドを受け取れる状態になると，下記に示すようにEDプロンプト，＊がスクリーンに表示される．

```
A>ED MYFILE.TEX
     : *
```

ファイルの前のバージョンが現在のディスクに存在しない場合には，EDは自動的に新しいファイルを生成し，下記のメッセージを表示する．

```
NEW FILE
     : *
```

注：編集セッションを開始する場合には，前もってSTATコマンドを使ってディスク上の使用可能な領域の大きさを調べておかなければならない．ディスクの使用可能な領域が少なくとも編集を行なおうとするファイルと同じ大きさであるか，あるいはファイルに文字を追加しようとする場合には，そのファイルよりも大きいかどうかを確認しておかなければならない．ディスク，またはディレクトリが一杯になった場合，EDには制限された回復メカニズムだけが存在する．これらについては，"EDエラー・メッセージ" の節で説明する．

5.3　EDオペレーション

EDでは，メモリ・バッファを通じてファイルの一部を変更することができる．上記に示されているいずれかのコマンドを使ってEDを開始する場合，このメモリ・バッファは空の状態である．コマンドを入力すると，EDはソース・ファイル，たとえばMYFILE．TEXのセグメントをメモリ・バッファに読取り編集可能な状態にする．そのファイルが新しい場合には，編集を行なう前にファイルにテキストを挿入しなければならない．

編集を行なっている間，EDは編集されたテキストを一時作業ファイルMYFILE．$$$に書き込む．

編集を終了すると，EDはメモリ・バッファの内容を一時ファイルに書込み，その後ソース・ファイルの残りのテキストを書き込む．次にEDはソース・ファイルの名前をMYFILE．TEXからMYFILE．BAKに変更するので，ユーザは必要に応じてバックアップ・ファイルからもとの内容を要求することができる．

次にEDは，一時ファイルMYFILE．$$$の名前を新しく編集されたファイルであるMYFILE．TEXに変更する．次の図は，ソース・ファイル，一時作業ファイルそれに新しいファイルの関係を示したものである．

注： 2つのファイル記述，つまり入力ファイルと出力ファイルの指定されたEDを入力すると，EDは入力ファイル名の型の部分を．BAKに変更しない．したがって，出力ファイルが別のディスクに書込まれる場合，入力ファイルは読取り専用，つまり書込み保護されたディスクに存在することとなる．

図5-1　EDオペレーション

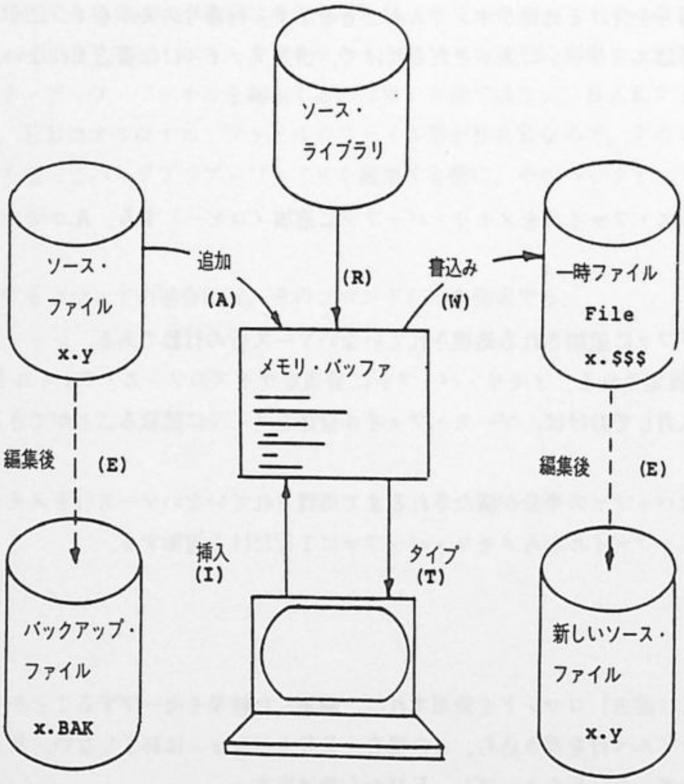


図5-1．でメモリ・バッファは論理的にソース・ファイルと一時作業ファイルの間に存在する．EDはソース・ファイル，メモリ・バッファ，一時ファイル，そして最終ファイルの間でテキスト行を転送するいくつかのコマンドをサポートする．次の表は3つの基本的なテキスト転送コマンドを示したものであり，これらのコマンドを使用すれば，容易にエディタに入り，一時ファイルにテキストを書込み，エディタから抜け出すことができる．

表5-1　テキスト転送コマンド

コマンド	結　　果
nA	ソース・ファイルの処理されていない次のnソース行をメモリ・バッファの最後に追加する.
nW	メモリ・バッファの最初のn行を一時ファイルの使用可能な領域に書込む.
E	編集を終了する. バッファのテキストはすべて一時ファイルにコピーされ, 処理されていないソース行もすべて一時ファイルにコピーされる. 次にファイル名が変更される.

5.3.1　テキストのバッファへの追加

EDを開始すると, メモリ・バッファは空の状態なので, A（追加）コマンドを使ってテキストをメモリ・バッファに追することができる.

注：EDでは, メモリ・バッファのデータを追跡できるようにテキストに行番号を付けることができる. EDを開始するときに表示されるコロンは, 行番号を付ける処理がオンであることを示す. 行番号の表示をオンにするには, EDプロンプトの後に−Vをタイプする. 行番号はスクリーンに表示されるだけで, 出力ファイルには書込まれない.

A（追加）コマンド

Aコマンドは, 既存のソース・ファイルをメモリ・バッファに追加（コピー）する. Aコマンドの形式は, 次の通りである.

```
nA
```

ただし, nはメモリ・バッファに追加される処理されていないソース行の行数である. シャープ記号（#）がnの代わりに指定されると, 整数65535が仮定される. メモリ・バッファに妥当なサイズのソース・ファイルを含むことができるので, 編集の始めにコマンド#Aを入力しておけば, ソース・ファイル全体をメモリに読取ることができる.

nが0の場合には, EDはバッファの半分が満たされるまで処理されていないソース行をメモリ・バッファに読取る. nを指定しないと, EDはソース・ファイルからメモリ・バッファに1行だけを追加する.

5.3.2　EDからの退去

W（書込み）コマンドとE（退去）コマンドを使用すれば, 編集した結果をセーブすることができる. Wコマンドは, メモリ・バッファから新しいファイルへ行を書き込む. この場合, EDセッションは終了しない. Eコマンドはバッファの内容とソース・ファイルの処理されていない行をセーブし, EDから抜け出す.

W（書込み）コマンド

Wコマンドは, バッファから新しいファイルへ行を書き込む. Wコマンドの形式は次の通りである.

```
nW
```

ただし, nはバッファの始めから新しいファイルの終りに書込まれる行の行数である. nが0より大きい場合には, EDはバッファの始めから新しいファイルの終りにn行を書込む. nが0の場合には, EDはバッファの半分が空の状態になるまで行を書込む.

ソース・ファイルからさらに行を読取るためにメモリ・バッファに領域が必要な場合には，ＯＷコマンドを使用すると便利である．下記に示すように，バッファの使用可能な領域の大きさを調べるには，ＯＶコマンドを実行するとよい．

```
1: *0V
25000/30000
1: *
```

上記のメッセージは，メモリ・バッファの全体のサイズが30,000バイトであり，使用可能な領域が25,000バイトあることを示している．

注：Ｗコマンドを実行したら，Ｈコマンドを入力して現在の編集セッションの間にセーブした行を再び編集しなければならない．

Ｅ（退去）コマンド

Ｅコマンドは E Dから正常に抜け出す．Ｅコマンドの形式は次の通りである．

```
E
```

この後にキャリッジ・リターンを入力する．

Ｅコマンドを入力すると，ＥＤはまずすべてのデータ行をバッファおよびソース・ファイルから新しいファイルに書込む．．ＢＡＫというファイルが存在する場合には，ＥＤはそれを削除し，オリジナル・ファイルのファイル型を．ＢＡＫに変更する．最後にＥＤは新しいファイル名をファイル名．．＄＄＄からもとのファイル型に変更し，制御をＣＣＰに戻す．

Ｅコマンドを使ってバック・アップ・ファイルを編集するのは賢い方法ではない．ＢＡＫファイルを編集し，Ｅコマンドを使ってＥＤから抜け出すと，ＥＤはオリジナル・ファイルのファイル型がＢＡＫなので，そのファイルを消去する．このような状況を避けるには，ＥＤを使ってバックアップ・ファイルを編集する前に，そのバックアップ・ファイルに他のファイル型を割当てておかなければならない．

注：ＥＤセッションを終了するコマンドの場合には，そのコマンドだけを指定する．

5.4 基本的な編集コマンド

今まで説明したテキスト転送コマンドを使用すれば，容易にエディタに入り，また抜け出すことができる．この節ではファイルを編集するための基本的なコマンドについて説明する．

ＥＤは，ファイルを文字の長い連鎖であるとして取扱う．ＥＤは文字ポインタ（ＣＰ）と呼ばれる概念上の装置をもとに文字および行を表示し，編集する．編集セッションでは，メモリ・バッファの中のＣＰの位置を頭の中に描いてＣＰを移動しファイルを編集するためのコマンドを入力しなければならない．

次のコマンドは，文字ポインタを移動したり，ＣＰの位置をもとにテキストを表示したりするコマンドである．これらのＥＤコマンドは数値引数と１文字のコマンドから構成されており，その後にキャリッジ・リターンを入力しなければならない．数値引数ｎは，ＥＤがそのコマンドを実行する回数を指定する．しかし，数値引数に関しては次の４通りの特殊な場合について考慮しなければならない．

・ 数値引数が省略されている場合には，ＥＤは引数が１であるとみなす．

・ コマンドがメモリ・バッファの逆方向に向かって実行される場合には，負の数値を使用する（Ｂコマンドは例外である）．

・ シャープ記号＃を数値の代わりに入力すると，ＥＤは引数として65535という値を使用する．シャープ記号の引数を符号の前に指定すれば，このコマンドはメモリ・バッファを逆方向に実行する（－＃）．

・ ＥＤが０を数値引数として受取るのは，特定のコマンドだけである．場合によっては，０を入力するとコマンドは可能な回数の約半分だけを実行するが，その他の場合にはＣＰが移動しないようにする．

下記の表は，基本的な編集コマンドとその有効引数をアルファベット順に要約したものである．

表5-2 基本的な編集コマンド

コマンド	動　作
B，－B	ＣＰをメモリ・バッファの始め（Ｂ）または終り（－Ｂ）に移動する．
nC，－nC	ＣＰをメモリ・バッファのｎ文字分だけ前へ（nC）またはｎ文字分だけ後ろへ(－nC）移動する．
nD，－nD	ＣＰの前のｎ文字（－nD）または後のｎ文字（ｎD）または後のｎ文字（ｎD）を削除す
I	挿入モードに入る．
I string↑Z	文字ストリングを挿入する．
nK，－nK	ＣＰの前のn 行（－ｎK）またはＣＰの後のｎ行（nK）を削除する．
nL，－nL	ＣＰをメモリ・バッファの前へｎ行（nL）またはｎ行後ろへ（－nL）移動する．
nT，－nT	ＣＰの前のｎ行（－nT）またはＣＰの後のｎ行（nT）をタイプする．
n，－n	ＣＰよりｎ行前へ（－ｎ）またはＣＰよりｎ行後ろへ（ｎ）移動し，その行を表示する．

この後の各節では，EDの基本的な編集コマンドをさらに詳しく説明する．これらの節に示されている例は，各コマンドがメモリ・バッファの文字ポインタの位置にどのように影響するかを示したものである．"EDコマンドの組合せ"という節に示されている例では，コマンドがスクリーンにどのように表示されるかが示されている．しかし，これらの節では，コマンド例の記号^は文字ポインタを示しており，メモリ・バッファの中に文字ポインタがどのように存在するかを想像しなければならない．

5.4.1 文字ポインタの移動

この節では，文字ポインタを移動するコマンドについて説明する．これらのコマンドでは目的行は表示されない．EDは基本的にプログラム・ソース・ファイルを作成し，編集するために使用するものであるが，EDを容易に理解できるように，以後の各節では簡単なテキスト例が示されている．

B（バッファの先頭／最後）コマンド

Bコマンドは，CPをメモリ・バッファの先頭または底部に移動する．Bコマンドの形式は次の通りである．

```
B, -B
```

－BはCPをメモリ・バッファの底部に移動する．BはCPをバッファの先頭に移動する．

C（文字）コマンド

Cコマンドは，CPを指定されている文字数だけ前または後へ移動する．Cコマンドの形式は次の通りである．

```
nC, -nC
```

ただし，nはCPが移動される文字数である．

正の数が指定されている場合には，CPは行の終りつまりバッファの底部へ向かって移動する．負の数値が指定されている場合には，CPは行の始めつまりバッファの先頭へ向かって移動する．nに十分大きな値を入力すれば，異なる行にCPを移動することができる．しかし，各行は目に見えない2つの文字によって区切られている．この文字は，キャリッジ・リターンとライン・フィードで，それぞれ<cr> <lf>によって表現される．したがって移動する文字数の中には，これらの文字も含まなければならない．たとえば，コマンド30Cを入力すると，CPは次の行に移動する．

```
Emily Dickinson said,<cr><lf>
"I fin^d ecstasy in living -<cr><lf>
```

L（行）コマンド

Lコマンドは，CPを指定されている行数だけ移動する．Lコマンドを実行した後，CPは常にその行の始めにセットされる．Lコマンドの形式は次の通りである．

```
nL, -nL
```

ただし，nはCPが移動する行数である．正の数を入力すると，CPはバッファの終りに向かって移動する．負の数を入力すると，CPはバッファの先頭に向かって移動する．コマンド2Lを入力すると，CPはメモリ・バッファの前へ2行分だけ移動し，文字ポインタはその行の始めを示す．

```
Emily Dickinson said,<cr><lf>
"I find ecstasy in living -<cr><lf>
^the mere sense of living<cr><lf>
```

CPはもともと行の中間にある文字を示しているが，コマンド-Lを実行すると，CPは前の行の始めにセットされる．特殊文字0を使用すれば，CPを現在の行の始めにセットすることができる．

n（行数）コマンド

nコマンドはCPを移動し，目的行を表示する．nコマンドの形式は次の通りである．

```
n, -n
```

ただし，nはCPが移動される行数である．このコマンドに対する応答として，EDは指定されている行数だけCPを前たは後ろへ移動し，目的行だけを印字する．

```
 Emily Dickinson said,<cr><lf>
^"I find ecstasy in living -<cr><lf>
```

このコマンドをさらに短縮し，数値をまったく入力しない形式をとることができる．コマンドをまったく入力しないでキリッジ・リターンを入力すると，それに対する応答としてEDはnコマンドのnが1であると仮定し，CPを次の行に移その行を印字する．

```
 Emily Dickinson said,<cr><lf>
^"I find ecstasy in living -<cr><lf>
```

また，数値を入力せずに負符号－だけを入力すると，CPは1行前の行にもどる．

5.4.2 メモリ・バッファの内容の表示

EDはテキストのどの部分を表示するか指定しない限り，メモリ・バッファの内容を表示しない．Tコマンドは，CPをかさずにテキストを表示するコマンドである．

T（タイプ）コマンド

Tコマンドは，CPの位置から指定された行数だけスクリーンに表示する．Tコマンドの形式は次の通りである．

```
nT, -nT
```

ただし，nは表示される行数を指定する．負の数が入力されている場合には，EDはCPの前のn行を表示する．正の入力されると，CPの後のn行が表示される．数値がまったく指定されていない場合には，EDは文字ポインタの位置からの行の終りまでを表示する．何行表示した場合でも，CPはもとの位置にセットされている．たとえば文字ポインタがメモ・バッファの先頭にあり，EDに対して4行を表示するように指示した場合（4T），スクリーンにその4行が表示されるがCPは行1の始めにセットされたままである．

```
^Emily Dickinson said,<cr><lf>
 "I find ecstasy in living -<cr><lf>
 the mere sense of living
 is joy enough."
```

CPが行の中間にある場合，数値の指定されていないTコマンドを入力すると，CPの位置からその行の終りまでの文表示される．しかし，文字ポインタは下記の例に示されているように，メモリ・バッファのもとの位置にセットされたまある．

```
"I find ec^stasy in living -
```

EDがTコマンドによってテキストを表示する場合には，常にCTRL-Sを入力することによりその表示を停止することできる．表示（スクローリング）を再び再開するにはCTRL-Q，表示を打切るにはCTRL-Cを入力する．

5.4.3 文字の削除

D（削除）コマンド

Ｄコマンドは指定された文字数を削除する．Ｄコマンドの形式は次の通りである．

```
nD, -nD
```

ただし，ｎは削除したい文字の文字数である．数値がまったく指定されていない場合には，ＥＤはＣＰの右側にある１文字だけを削除する．正の数が入力された場合には，ＣＰの右側にある複数の文字をファイルの終りに向かって削除する．負の数値が入力された場合には，ＣＰの左側にある文字をファイルの始めに向かって削除する．文字ポインタが下記のようにメモリ・バッファにセットされている場合，

```
Emily Dickinson said,<cr><lf>
"I find ecstasy in living -<cr><lf>
the mere sense of living<cr><lf>
is joy ^enough."<cr><lf>
```

コマンド６ＤはＣＰの後の６文字を削除し，その結果，メモリ・バッファは次のようになる．

```
Emily Dickinson said,<cr><lf>
"I find ecstasy in living -<cr><lf>
the mere sense of living<cr><lf>
is joy ^."<cr><lf>
```

またＤコマンドを使用すれば，２つの行の間にある〈ｃｒ〉〈ｌｆ〉を削除し，その２行を１行に結合することができる．〈ｃｒ〉と〈ｌｆ〉は２文字であることを忘れてはならない．

K（消去）コマンド

Ｋコマンドはメモリ・バッファの行全体を〝消去〟つまり削除する．Ｋコマンドの形式は次の通りである．

```
nK, -nK
```

ただし，ｎは削除される行数である．正の数が入力されると，ＣＰの後のｎ行が削除される．負の数が入力されていれば，ＣＰの前のｎ行が削除される．数値がまったく指定されていない場合には，ＥＤは現在の行を削除する．文字ポインタが２番目の行の始めにセットされている場合には（下記のように），

```
 Emily Dickinson said,<cr><lf>
^"I find ecstasy in living -<cr><lf>
 the mere sense of living<cr><lf>
 is joy enough."<cr><lf>
```

コマンド-Ｋは前の行を削除し，その結果メモリ・バッファは次のようになる．

```
^"I find ecstasy in living -<cr><lf>
 the mere sense of living<cr><lf>
 is joy enough."<cr><lf>
```

ＣＰが行の途中にある場合には，ＫコマンドはＣＰからその行の終りまでの文字だけを削除し，ＣＰの前の文字と次の行を連結する．

－Ｋコマンドは前の行の始めとＣＰの間にある文字をすべて削除する．

０Ｋコマンドはその行の始めからＣＰまでの間に含まれる文字を削除する．

特殊文字＃を使用すると，ＣＰの位置からバッファの先頭または終りまでの間に含まれるすべてのテキストを削除することができる．＃Ｋを使用する場合には，メモリ・バッファから削除した行を再び要求することができないので十分注意する必要がある．

5.4.4 メモリ・バッファへの文字の挿入

I（挿入）コマンド

文字をスクリーンからメモリ・バッファに挿入するには，Ｉコマンドを使用する，Ｉコマンドの形式は次の通りである．

```
I
Istring^Z
```

最初のコマンドをタイプすると，ＥＤは挿入モードに入る．このモードでは，入力されたデータはすべて直接メモリ・バ
ファに追加される．ＥＤはそれらの文字をその行に入力し，ＥＮＴＥＲキーを押すまで新しい行を開始しない．

```
A>ED B:QUOTE.TEX

NEW FILE
   : *i
  1:  Emily Dickinson said,
  2:  "I find ecstasy in living -
  3:  the mere sense of living
  4:  is joy enough."
  5:  ^Z
   : *
```

注：挿入モードから抜け出すには，ＣＴＲＬ-ＺまたはEscを入力しなければならない．ＥＤプロンプト＊がスクリーンに示された場合，ＥＤは挿入モードではなくなる．

コマンド・モードでは，ＣＰ／M-86の行編集制御文字を使って入力を編集することができる．下記の表は，これらの文字を示したものである．

表5-3　ＣＰ／M-86の行編集制御文字

コマンド	結　　果
ＣＴＲＬ-Ｃ	エディタを中断し，ＣＰ／M-86システムにもどる．
ＣＴＲＬ-Ｅ	コマンド行をバッファに書込まずに，長い行に対し，キャリッジ・リターンを行なう．
ＣＴＲＬ-Ｈ	現在行の最後にタイプされた文字を１文字だけ削除する．
ＣＴＲＬ-Ｕ	現在タイプしている行全体を削除する．
ＣＴＲＬ-Ｘ	現在タイプしている行全体を削除する．ＣＴＲＬ-Ｕと同じである．
Rubout	最後の文字を削除し，削除された文字をスクリーンに表示する．

注：挿入モードでは，ＣＴＲＬ-ＣおよびＣＴＲＬ-Ｅを除いて同じ行編集制御が存在する．

数値と英字を組合わせて入力する場合，端末装置がシフト・インの数値を特殊文字に変換するとすれば，シフト・イン
ーを使用することは不便である．

ＥＤは，数字に影響を与えずに英字入力を大文字に変換する２通りの方法を規定している．その最初の方法は，挿入コ
ドの文字を大文字：Ｉで入力することである．大文字によるコマンド入力の場合，挿入モードやストリングとして入力さ
英字は，すべて大文字に変換される．（挿入コマンド文字を小文字で入力すると，英字はすべてタイプされた通りに挿入さ
る）．２番目の方法は，テキストを挿入する前にＵコマンドを入力する方法である．この場合，－Ｕコマンドが入力され
で大文字への変換が行なわれる．

Iストリング^Z（ストリングの挿入）コマンド

Iコマンドの2番目の形式は，挿入モードに入るためのコマンドではない．このコマンドは文字ストリングをメモリ・バッファに挿入し，直ちにEDプロンプトにもどる．コマンド・ストリングの編集には，CP／M-86行編集制御文字を使用することができる．

ストリングを挿入するには，まずCPの位置をセットするコマンドを使用する．ストリングを挿入したい位置にCPを移動しなければならない．たとえば，ストリングを最初の行の始めに挿入したい場合には，Bコマンドを使ってCPをバッファの先頭に移動しなければならない．CPの位置が正しくセットされたら，下記のようにストリングを入力する．

```
iIn 1870, ^Z
```

このコマンドは，最初の行の始めに〝In 1870〟というフレーズを挿入し，直ちにEDプロンプトにもどる．このストリングが挿入された後，メモリ・バッファは次のようになる．

```
In 1870, ^Emily Dickinson said,<cr><lf>
```

5.4.5　文字の置換

S（置換）コマンド

Sコマンドはメモリ・バッファを調べ，指定されたストリングを見つけるが，そのストリングが見つかったら自動的に新しいストリングと置換える．Sコマンドの形式は次の通りである．

```
nSsearch string^Znew string
```

ただし，nは置換を行なう回数を指定する．nが指定されていない場合には，EDはメモリ・バッファ中でサーチ・ストリングと最初に対応するものを見つける．たとえば次のコマンドは，

```
Emily Dickinson^ZThe poet
```

〝Emily Dickinson〟というストリングに対応する最初のストリングを捜し，その代わりに〝The poet〟を挿入する．CPの位置は置換られたフレーズの後にセットされ，メモリ・バッファの内容は下記のようになる．

```
The poet^ said,<cr><lf>
```

大文字のSコマンドによって大文字への変換がなされる場合には，EDはサーチ・ストリング，挿入ストリングとも大文字に変換する．このコマンドを他のコマンドと組合わせる場合には，CTRL-Zで新しいストリングを終了しなければならない．

5.5 EDコマンドの組合わせ

キーストロークと編集時間を節約するには，編集および表示コマンドを組合わせるとよい．EDコマンドは，同じ行にいくつでもタイプすることができる．EDはユーザがキャリッジ・リターン・キーを押した後で，初めてコマンド・ストリングを実行する．CP／M-86の行編集制御を使用して，EDコマンド・ストリングを取扱う．

複数のコマンドを1行に結合する場合，EDは入力された順序で左から右に向かってそれらのコマンド行を実行する．EDコマンドを組合わせるには，次の4つの制約事項がある．

- EDコマンドを組合わせたコマンド行が，CP／M-86の1行の最大文字数である128文字を越えてはならない．
- EDコマンドを組合わせたコマンド行に文字ストリングが含まれる場合には，その行は100文字を越えてはならない．
- 編集セッションを終了するためのコマンドを，EDコマンドを組合わせたコマンド行に指定してはならない．
- I，S，J，X，Rコマンドなどのように文字ストリングやファイル記述を必要とするコマンドは，文字ストリングファイル記述がまったく指定されない場合でも，その行の最後に入力するか，あるいはCTRL-ZまたはESC文字によって終了しなければならない．

前の節の例は，メモリ・バッファと文字ポインタの位置を示しているが，この節では編集セッションの間にスクリーンにどのような情報が表示されるかを示している．文字ポインタは概念上のものであるが，EDコマンドは文字ポインタの位置をもとにテキストを表示したり，編集したりするので，ユーザは常に文字ポインタの位置を頭の中で描いておかなければならない．

5.5.1 文字ポインタの移動

文字数を計算せずにCPをある行の終りに移動するには，LコマンドとCコマンドを組合わせ，L-2Cを入力する．このコマンド・ストリングは行の終りにある〈cr〉〈lf〉を計算に入れて処理を行なう．

このコマンド・ストリングのCコマンドを変更すれば，CPをさらに左側に移動することができる．行の終りを変更しなければならない場合，変更する前に文字数を計算したくないときには，下記の例に示すようにこのコマンドを使用することができる．

```
          1: *T
          1:  Emily Dickinson said,
          1: *L-7CT
     said,
          1: *
```

5.5.2 テキストの表示

Tコマンドは，CPから行の終りまでをタイプする．行全体を表示したい場合には，LコマンドとTコマンドを組合わせOLtを入力するとCPは行の途中から行の始めに移動し，その行全体が表示される．

下記の例ではCPは行の途中にある．OLを入力することにより，CPはその行の始めに移動し，TはCPからその行の終りまでをタイプするので，行全体を表示することができる．

```
          3: *T
     sense of living
          3: *olt
          3:  the mere sense of living
          3: *
```

コマンドOTTはCPを移動せずに，行全体を表示する．

ＥＤコマンドがＣＰを正しく移動したかどうかを調べるには，コマンドとＴコマンドを組合わせ，その行を表示する．下記の例では，ＣコマンドとＴコマンドが組合わされている．

```
    2: *8CT
ecstasy in living -
    2: *
```

```
    4: *B#T
    1: Emily Dickinson said,
    2: "I find ecstasy in living -
    3: the mere sense of living
    4: is joy enough."
    1: *
```

5.5.3 編集

テキストを編集し，編集した結果をすみやかに調べるには，編集コマンドと他のＥＤコマンドを組合わせ，ＣＰを移動し，テキストを表示する．下記のようなコマンド・ストリングは，ＣＰを移動し，指定された文字を削除して変更された結果を直ちに調べるためのコマンドである．

```
1: *15C5D0LT
1:  Emily Dickinson,
1: *
```

編集とコマンドＫと他のＥＤコマンドを組合わせると，行全体を削除し，編集結果を直ちに調べることができる．

```
1: *2L2KB#T
1:  Emily Dickinson said,
2: "I find ecstasy in living -
1: *
```

Ｉ（挿入）コマンドの短縮形を入力すると，テキストを簡単に変更することができる．これらの変更を行ないその結果を調べるには，Ｉコマンド・ストリングとＣコマンド，それに０ＬＴコマンド・ストリングを下記のように組合わせる．挿入ストリングはＣＴＲＬ-Ｚで終了しなければならない．

```
1: *20Ci to a friend^Z0LT
1:  Emily Dickinson said to a friend,
1: *
```

5.6 拡張EDコマンド

今までに説明した基本的な編集コマンドを使用すれば，EDを使ってあらゆる編集を行なうことができる．しかし，次のEDコマンドを使用すれば，EDの機能をさらに高めることになる．

5.6.1 CPの移動とテキストの表示

P（ページ）コマンド

Tコマンドを使用すれば，テキストをどれだけでも表示できるが，バッファを〝ページ区切り〟した方が便利な場合もある．ページ区切りとは1画面分のデータを表示し，同時にCPを各画面の始めに移動することである．このような処理を行なうには，EDのPコマンドを使用する．Pコマンドの形式は下記の通りである．

```
nP, -nP
```

ただし，nは表示したいページのページ番号である．nを指定しなければ，EDはCPの後の23行をタイプし，CPを23行だけ先に進める．この時，CPは画面の1文字目にセットされる．

CPを移動せずに現在のページを表示するには，OPを入力する．特殊文字0はCPが移動しないようにする文字である．nに対して負の数を指定すると，Pはファイルの始めに向かって前の方へページ区切りを行なう．

n：(行番号）コマンド

行番号が表示されている場合，コマンドの一部として行番号を入力してCPの宛先を指定することができる．行番号コマンドの形式は次の通りである．

```
n:
```

ただし，nは宛先行の行番号である．このコマンドはCPを指定された行の始めにセットする．たとえばコマンド4：を入力すると，CPは4行目の始めにセットされる．

EDは，行が追加されたり削除されたりするたびに，バッファ中のテキスト行の行番号を動的に変更する．したがって，編集中に宛先行の行番号は変更される可能性がある．

：n（実行開始行番号）コマンド

行番号コマンドの形式を逆にしたコマンドは，特定の行番号からコマンドが実行されることを指定する．このコマンドを使用できるのは，次の3つのEDコマンドと組合わせた場合だけである．T（タイプ）コマンド，L（行）コマンド，K(キル）コマンド，：nコマンドの形式は次の通りである．

```
:ncommand
```

ただし，nはコマンドの実行が開始される行番号である．コマンドの：nの部分はCPを移動しないが，その後に続くコマンドがCPを移動するかもしれない．

n：と：nを組合わせ，コマンドの実行を開始したい行を広範囲にわたって指定することができる．たとえば2：：4Tというコマンドを入力すると，下記に示すように，2，3，4行目がそれぞれ表示される．

```
1: *2::4T
2:  "I find ecstasy in living -
3:  the mere sense of living
4:  is joy enough."
2: *
```

5.6.2 文字ストリングの探索と置換

ＥＤはメモリ・バッファ全体をサーチし，ＣＰを希望するワードまたはフレーズの後にセットするコマンドＦをサポートしている．

Ｎコマンドを使用すれば，ＥＤはバッファではなく，ソース・ファイル全体をサーチする．Ｊコマンドを使用すれば，文字ストリングがサーチされ，置換えられる．

Ｆ（サーチ）コマンド

Ｆコマンドは最も簡単なサーチ機能を実行する．その形式は次の通りである．

```
nFstring
```

ただし，ｎはストリングをサーチする回数である．ＥＤはＣＰからバッファの終りに向かってだけしかサーチできないので，ｎは正の数でなければならない．数値をまったく入力しないと，ＥＤは指定されたストリングに対応するファイルの中の最初のストリングを見つける．次の例では，ワードlivingに対応するストリングの中で２番目のストリングが見つけられる．

```
1: *2fliving
3: *
```

文字ポインタは，ワード〝living〟に対応する２番目のストリングが存在する３行目の始めに移動する．その行を表示するには，サーチ・コマンドとタイプ・コマンドを組合わせる．Ｆコマンドの後に別のＥＤコマンドを続ける場合には，下記に示すように，ストリングの最後にＣＴＲＬ-Ｚを入力しなければならない．

```
1: *2fliving^Z0lt
3: *the mere sense of living
```

Ｆコマンドを大文字で入力するか，小文字で入力するかによって結果に違いが生じる．Ｆを入力すると，ＥＤは内部的に引数ストリングを大文字に変換する．ｆを指定すると，ＥＤは入力された通りのストリングに正しく対応するストリングを見つける．たとえば，ＦＣＰ／m-86を入力すると，ＣＰ／M-86がサーチされるが，fCp/m-86 を入力するとＣp／m-86はサーチされるが，ＣＰ／M-86またはcp／m-86を見つけることはできない．

ＥＤがメモリ・バッファから対応するストリングを見つけることができない時には，次のメッセージを出力する．

```
BREAK "#" AT
```

記号#は，Ｆコマンドの実行中にサーチが失敗したことを示す．

Ｎコマンド

Ｎコマンドは，サーチ機能をメモリ・バッファだけではなく，ソース・ファイルにまで拡張したものである．サーチが成功すると，サーチされたストリングの最初の文字にＣＰをセットする．Ｎコマンドの形式は次の通りである．

```
nNstring
```

ただし，ｎは対応するストリングの中で何番目のストリングをサーチしたいかを示す．数値がまったく入力されていない場合には，ＥＤはファイルのそのストリングに対応する最初のストリングを見つける．Ｎコマンドが大文字で入力されるか小文字入力されるかは，Ｆコマンドの場合と同じような効果を持たらす．Ｎコマンドの後に他のＥＤコマンドを指定する場合には，ストリングの最後にＣＴＲＬ-Ｚを入力しなければならない．

Ｎコマンドを実行すると，ＥＤはメモリ・バッファから指定されたストリングをサーチする．しかしＥＤがそのストリングを見つけることができなくても，エラー・メッセージは出力されない．その代わり，ＥＤは自動的にサーチされたデータをバッファから新しいファイルに書込む．次にＥＤは０Ａコマンドを実行し，ソース・ファイルからまだサーチされていないデータをバッファに読取る．ＥＤはバッファのサーチを続け，指定されたストリングが見つかるまで，あるいはソース・ファイルの終りに到達するまでサーチしたデータを新しいファイルに書込み，新しいデータをソース・ファイルから読取る．ＥＤがソース・ファイルの終りに到達すると，次のメッセージが出力される．

```
BREAK  "#" AT
```

EDはサーチされたデータを新しいファイルに書込んでから，ソース・ファイルから次のデータを読取るので，ソース・ファイルの終りに到達し，エラー・メッセージが出力される前に，常にバッファの内容は新しいファイルに書込まれている．

注：ソース・ファイルの終りに到達し，メモリ・バッファが空の状態になった後，編集セッションを続けるには，Hコマンドを使用しなければならない．

J（並置）コマンド）

Jコマンドはサーチ・ストリングの後にストリングを挿入し，挿入されたストリングの終りから第3引数の〝delete-to〟（削除したい）ストリングの始めまでに含まれる文字をすべて削除する．このコマンドは，サーチ・ストリングと削除したいストリングの間のストリングを，挿入ストリングと置換える．

Jコマンドの形式は次の通りである．

```
nJsearch string^Zinsert string^Zdelete-to string
```

nは，サーチ・ストリングと対応するストリングの中で，何番目のストリングを見つけるかを示すものである．数値がまったく指定されていない場合には，EDはメモリ・バッファの中でサーチ・ストリングに対応する最初のストリングを見つける．

下記の例ではEDはワード〝Dickinson〟をサーチし，その後にフレーズ〝told a friend〟を挿入し，次にカンマまでの間に含まれるすべてを削除する．

```
1: *#T
1:  Emily Dickinson said,
2:  "I find ecstasy in living -
3:  the mere sense of living
4:  is joy enough."
1: *jDickinson^Z told a friend^Z,
1: *0lt
1:  Emily Dickinson told a friend,
1: *
```

このコマンドを他のコマンドと組合わせる時には，削除したいストリング（delete-to）の最後にCTRL-Z または Escを入力しなければならない．（これは下記の例に示されている）．大文字のJコマンドを入力すると，EDはサーチ・ストリングおよび削除ストリングを大文字に変換し，また大文字に変換した挿入ストリングを挿入する．

Jコマンドが特に役立つのは，下記のようにアセンブリ言語ソース・コードの注釈を変更する場合である．

```
236:  SORT    LXI     H, SW     ;ADDRESS TOGGLE SWITCH
236: *j;^ZADDRESS SWITCH TOGGLE^Z^L^Z0LT
236:  SORT    LXI     H, SW     ;ADDRESS SWITCH TOGGLE
236: *
```

この例では，EDは最初のセミコロンをサーチし，その後にADDRESS SWITCH TOGGLEを挿入し，次にCTRL-Lで表現されている〈cr〉〈lf〉までを削除する．（どのサーチ・ストリングでも，希望するフレーズが複数の行にまたがっている場合には，CTRL-Lを使用することにより〈cr〉〈lf〉を表現することができる．また，サーチ・ストリングの中でCTRL-Iを使用すれば，タブを表現することができる）．

注：長いストリングを使用する場合，コマンドの長さがスクリーンの長さよりも長くなる場合には，CTRL-Eを入力すると，画面上では物理的なキャリッジ・リターンが行なわれる．CTRL-Eはカーソルを画面の左端にもどすが，コマンド行はEDに送られない．ストリングを含むEDコマンド行の最大文字数は100文字である．コマンドを終了する場合には，キャリッジ・リターン・キーを押し，コマンドをEDに送らなければならない．

M（マクロ）コマンド

EDマクロ・コマンド　Mは，コマンド・ストリングの機能をさらに拡張する．Mコマンドを使用すれば，実行を繰返し行なうためにEDコマンドをグループ化することができる．

Mコマンドの形式は次の通りである．

```
nMcommand string
```

ただし，nはコマンド・ストリングが実行される回数である．Mコマンドの引数として，負の数値を入力することはできない．数値がまったく指定されていない場合には，特殊文字#であると仮定され，ストリングの中に指定されているコマンドに応じて，バッファの中のデータの終りまたはソース・ファイルの終りに到達するまで，コマンド・ストリングを実行する．

下記の例では，EDはメモリ・バッファの終りに到達するまで4つのコマンドを繰返し実行する．

```
        1: *mfliving^Z-6diLiving^Z0lt
        2:  "I find ecstasy in Living -
        3:  the mere sense of Living

    BREAK "#" AT ^Z
  3: *
```

Mコマンドの最後にはキャリッジ・リターンを入力する．したがって，Mコマンドはその行の最後のコマンドでなければならない．また，マクロに指定されるすべての文字ストリングは，CTRL-ZまたはEscによって終了しなければならない．文字ストリングがコマンドを組合わせたストリングで終了する場合には，CTRL-Zで終了し，その後に〈cr〉を入力することによってMコマンドを終了しなければならない．

Mコマンドは，マクロの実行される回数を制限しておけば，EDがバッファの終りまたはソース・ファイルの終りに到達しない場合でも，BREAK "#" というメッセージで終了する．通常，メッセージの中に表示されるコマンド文字はMではなく，ストリングの中のコマンドの1つである．

マクロ・コマンドを途中で中断するには，キーボードからCTRL-Cを入力する．

5.6.3 テキスト・ブロックの移動

あるデータ領域から別の領域へ複数の行を移動するには，Xコマンドを使ってテキスト・ブロックを一時ファイル .LIBに書込み，それからKコマンドを使ってこれらの行をもとの位置から削除し，最後にRコマンドを使ってそのブロックを新しい位置に読取る．

Xコマンド

Xコマンドの形式は次の通りである．

```
nX
nX filespec^Z
```

nはCPからバッファの終りに向かっての行数であり，一時ファイルに書込まれる行を示す．したがって，nは常に正の数値でなければならない．ファイル名が指定されていない場合には，X$$$$$$$であると仮定される．ファイル型が指定されていない場合には，.LIBであると仮定される．Xコマンドがその行の最後のコマンドでない場合には，このコマンドの最後にCTRL-ZまたはEscを入力しなければならない．

下記の例では1行だけが一時ファイルに書込まれる．

```
1: *X
1: *t
1: *Emily Dickinson said,
1: *kt
1: *"I find ecstasy in living -
1: *
```

ライブラリ・ファイルが指定されていない場合，EDはX$$$$$$$.LIBという名前のファイルを見つける．ファイルが存在しない場合には，EDはそのようなファイルを作成する．前のXコマンドで既にライブラリ・ファイルが作成されている場合には，EDはそのファイルの終りに指定された行を追加する．

Xコマンドの中で引数nとして特殊文字0を使用すると，EDの中から任意のファイルを削除することができる．

R（読取り）コマンド

Xコマンドは現在の行から次のn行をライブラリ・ファイルに書込む．Rコマンドは書込まれた行を検索することができる．Rコマンドの形式は次の通りである．

```
R
Rfilespec
```

ファイル名が指定されていない場合には，X$$$$$$$であると仮定される．ファイル型が指定されていない場合には．.LIBであると仮定される．RはCPの前にライブラリ・ファイルを挿入する．したがって，ファイルがメモリ・バッファに追加された後，文字は新しい行番号上に存在するが，CPの位置は読取りを行なう前と同じ位置である．

Rコマンドと他のコマンドを組合わせた時には，下記の例のようにファイル名と後続のコマンドとの間をCTRL-Zで区切らなければならない．下記の例では，EDは読取った結果を調べるためにファイル全体を表示する．

```
1: *4l
 : *R^ZB#T
1:  "I find ecstasy in living -
2:  the mere sense of living
3:  is joy enough."
4:  Emily Dickinson said,
1: *
```

5.6.4 変更データの保管と放棄：EDからの退去

次の3つのコマンドを使用すれば，編集した結果をセーブしたり，放棄したりすることができる．

H（ファイルのヘッド）コマンド

Hコマンドは，EDセッションを終了せずにメモリ・バッファの内容をセーブするが，処理はファイルの"先頭"にもどされる．したがってHコマンドは現在の変更結果をセーブし，EDを終了せずにそのファイルを再編集する．Hコマンドの形式は次の通りである．

```
H
```

この後にキャリッジ・リターンを入力する．

Hコマンドを実行するために，EDはまず新しいファイルに対する処理を終了し，バッファの残りの行とソース・ファイルの残りの行をすべて新しいファイルに書込む．次にEDは新しいファイルをクローズし，もとのソース・ファイルと同じファイル記述を持つ.BAKというファイル型のファイルを消去し，もとのソース・ファイルのファイル型を.BAKに変更する．

次にEDはファイル型が.$$$である新しいファイルの名前をもとのファイル記述に変更する．最後にEDは新しく名前が変更されたファイルを，新たに編集するソース・ファイルとしてオープンし，新しい.$$$ファイルをオープンする．EDが*というプロンプトを出力したとき，CPの位置は空のメモリ・バッファの始めにセットされている．

編集結果をもとのファイル以外のファイルに書込みたい場合には，次のコマンドを使用する．

```
A>ED filespec differentfilespec
```

Hコマンドを使って編集を再開する場合には，その編集が終了した段階でEDはファイルdifferentfile name.$$$というファイルの名前をdifferentfile name.BAKに変更し，different filespecという新しいファイルを作成する．

O（オリジナル）コマンド

Oコマンドは編集セッションで行なわれた変更結果を無視するので，オリジナル・リース・ファイルがそのままもどされ，ユーザはEDセッションを終了せずに再び編集を行なうことができる．Oコマンドの形式は次の通りである．

O

このあとキャリッジ・リターンを入力する．Oコマンドを入力すると，EDは次のメッセージを表示することによって変更結果を無視してよいかどうかを質問する．

O (Y/N)?

ユーザはYまたはNを応答しなければならない．他のキーを入力すると，EDは同じ質問を繰返し表示する．Yを入力すると，EDは一時ファイルとメモリ・バッファの内容を消去する．＊というプロンプトが出力されたとき文字ポインタの位置はEDを開始したときと同じように，空のメモリ・バッファの始めにセットされている．

Q（中断）コマンド

Qコマンドは，EDセッションの始めからEDを抜け出すまで行なわれた変更結果を放棄する．Qコマンドの形式は次の通りである．

Q

この後にキャリッジ・リターンを入力しなければならない．

Qコマンドを入力すると，EDは次の質問を表示することにより変更結果を無視してよいかどうかを確認する．

Q (Y/N)?

ユーザはYまたはNを応答しなければならない．他のキーを入力すると，EDは同じ質問を繰返す．Yを入力するとEDは一時ファイルを消去し，ソース・ファイルをクローズし，制御をCP／M-86にもどす．

注：CTRL-Cを入力すれば，直ちに制御をCP／M-86にもどすことができる．このようにすれば，EDはソース・ファイルまたは新しいファイルをクローズすることができないので，一時ファイルは削除されない．

5.7 EDエラー・メッセージ

EDは，2種類のエラー・メッセージのいずれかを出力する．EDが編集コマンドを実行できない場合には，EDエラ メッセージが出力され，EDは指定されたファイルを読取ることができない．また，書込むことができない場合には，C M-86エラー・メッセージが出力される．

EDエラー・メッセージの形式は次の通りである．

```
BREAK "x" AT c
```

ただし，Xは次の表に定義されている記号のいずれかであり，エラーの発生したコマンド文字である．

表5-4 EDエラー記号

記号	意　味
#	サーチの失敗．EDがF，SまたはNコマンドで指定されているストリングを見つけることができなかった場合，このメッセージが出力される．
?C	コマンド文字Cを認識できない．EDが示されているコマンド文字を認識できないか，あるいはE，H，Qまたは0コマンドが他のコマンドと組合わされて指定されている．
O	.LIBファイルが存在しない．EDがRコマンドに指定されている .LIBファイルを見つけることができなかった．
>	バッファが一杯である．EDがメモリ・バッファに文字を追加できない場合や，F，MまたはSコマンドに指定されているストリングが長すぎる．
E	コマンド中断．キーボードから入力された文字によりコマンドの実行が途中で打切られると，このメッセージが出力される．
F	ファイル・エラー．DISK FULLまたはDIRECTORY FULLの後に出力される．

下記の例は，一般的な編集エラーからの回復方法を示したものである．たとえば，

```
BREAK ">" AT A
```

このメッセージは，Aコマンドの実行を終了する前にメモリ・バッファが一杯になったことを示す．このようなエラー生すると，文字ポインタはバッファの終りにセットされ，その後，編集処理を行なうことはできない．この場合には，0W マンドを使ってバッファの半分をファイルに書込むか，OまたはHコマンドを使ってそのファイルの編集をやり直す．

```
BREAK "#" AT F
```

このメッセージは，EDがFコマンドに指定されているストリングと対応するストリングを見つけることができず，メモ ・バッファの終りに到達したことを示す．このとき，文字ポインタの位置はバッファの終りにセットされている．この場 は，Bまたはn：行番号コマンドを使ってCPを移動し，編集を再開する．

```
BREAK "F" AT F
DISK FULL
```

この場合には，OXコマンドを使ってディスク上の不必要なファイルを消去するか，B#Xd：buffer .sav コマンドを使 メモリ・バッファの内容を他のディスクに書込む．

```
BREAK "F" AT n
DIRECTORY FULL
```

前のメッセージで説明したのと同じコマンドを使って，このファイル・エラーを回復する．

下記の表は，ＥＤがファイルの読取りまたは書込みできない場合に出力される，ディスク・ファイル・エラー・メッセージを定義したものである．

表5-5　ＥＤディスク・ファイル・エラー・メッセージ

メッセージ	意　　味
BDOS ERR ON d: RO	ディスクｄ：は読取り専用属性を持つ，最後にコールド・ブートまたはウォーム・ブートされた後でドライブに別のディスクが挿入された場合，このような状況が発生する．
** FILE IS READ ONLY **	ＥＤを呼び出すために指定されているファイルがＲＯ属性を持つ．ＥＤはユーザがそのファイルの内容を調べることができるようにファイルを読取ることができるが，読取り専用ファイルの内容を変更することはできない．

付録A

ＡＳＣＩＩと16進数の変換テーブル

ＡＳＣＩＩとはAmerican Standard Code for Information Interchange（情報交換用米国標準コード）の略称である．このコードには96文字の印刷可能な文字と，ディスクにデータを格納するために使用される32文字の印刷不可能な文字とが含まれている．

表Ａ－１はＡＳＣＩＩ記号を定義したものであり，表Ａ－２はＡＳＣＩＩと16進数の変換テーブルを示したものである．このテーブルには２進数，８進数，16進数およびＡＳＣＩＩの変換が示されている．

表Ａ－1．ＡＳＣＩＩ記号

記号	意味
ＡＣＫ	肯定応答
ＢＥＬ	ベル
ＢＳ	バックスペース
ＣＡＮ	取消し
ＣＲ	キャリッジ・リターン
ＤＣ	装置制御
ＤＥＬ	削除
ＤＬＥ	データ・リンク・エスケープ
ＥＭ	媒体の終り
ＥＮＱ	照会
ＥＯＴ	伝送の終り
ＥＳＣ	エスケープ
ＥＴＢ	伝送の終り
ＥＴＸ	テキストの終り
ＦＦ	フォーム・フィード
ＦＳ	ファイル・セパレータ
ＧＳ	グループ・セパレータ
ＨＴ	水平タブ
ＬＦ	ライン・フィード
ＮＡＫ	否定応答
ＮＵＬ	ヌル（空白文字）
ＲＳ	レコード・セパレータ
ＳＩ	シフト・イン

G記号	意　味
SO	シフト・アウト
SOH	見出しの始め
SP	スペース
STX	テキストの始め
SUB	代用
SYN	同期アイドル
US	ユニット・セパレータ
VT	垂直タブ

表A-2. ASCII変換テーブル

2進数	10進数	16進数	ASCII
0000000	0	0	NUL
0000001	1	1	SOH (CTRL-A)
0000010	2	2	STX (CTRL-B)
0000011	3	3	ETX (CTRL-C)
0000100	4	4	EOT (CTRL-D)
0000101	5	5	ENQ (CTRL-E)
0000110	6	6	ACK (CTRL-F)
0000111	7	7	BEL (CTRL-G)
0001000	8	8	BS (CTRL-H)
0001001	9	9	HT (CTRL-I)
0001010	10	A	LF (CTRL-J)
0001011	11	B	VT (CTRL-K)
0001100	12	C	FF (CTRL-L)
0001101	13	D	CR (CTRL-M)
0001110	14	E	SO (CTRL-N)
0001111	15	F	SI (CTRL-O)
0010000	16	10	DLE (CTRL-P)
0010001	17	11	DC1 (CTRL-Q)
0010010	18	12	DC2 (CTRL-R)
0010011	19	13	DC3 (CTRL-S)
0010100	20	14	DC4 (CTRL-T)
0010101	21	15	NAK (CTRL-U)
0010110	22	16	SYN (CTRL-V)
0010111	23	17	ETB (CTRL-W)
0011000	24	18	CAN (CTRL-X)
0011001	25	19	EM (CTRL-Y)
0011010	26	1A	SUB (CTRL-Z)
0011011	27	1B	ESC (CTRL-[)
0011100	28	1C	FS (CTRL-\)
0011101	29	1D	GS (CTRL-])
0011110	30	1E	RS (CTRL-^)
0011111	31	1F	US (CTRL-_)

2進数	10進数	16進数	ASCII
0100000	32	20	(SPACE)
0100001	33	21	!
0100010	34	22	"
0100011	35	23	#
0100100	36	24	$
0100101	37	25	%
0100110	38	26	&
0100111	39	27	'
0101000	40	28	(
0101001	41	29	)
0101010	42	2A	*
0101011	43	2B	+
0101100	44	2C	,
0101101	45	2D	−
0101110	46	2E	.
0101111	47	2F	/
0110000	48	30	0
0110001	49	31	1
0110010	50	32	2
0110011	51	33	3
0110100	52	34	4
0110101	53	35	5
0110110	54	36	6
0110111	55	37	7
0111000	56	38	8
0111001	57	39	9
0111010	58	3A	:
0111011	59	3B	;
0111100	60	3C	<
0111101	61	3D	=
0111110	62	3E	>
0111111	63	3F	?
1000000	64	40	@
1000001	65	41	A
1000010	66	42	B
1000011	67	43	C
1000100	68	44	D
1000101	69	45	E
1000110	70	46	F
1000111	71	47	G
1001000	72	48	H
1001001	73	49	I
1001010	74	4A	J
1001011	75	4B	K
1001100	76	4C	L
1001101	77	4D	M
1001110	78	4E	N
1001111	79	4F	O
1010000	80	50	P
1010001	81	51	Q

2進数	10進数	16進数	ASCII
1010010	82	52	R
1010011	83	53	S
1010100	84	54	T
1010101	85	55	U
1010110	86	56	V
1010111	87	57	W
1011000	88	58	X
1011001	89	59	Y
1011010	90	5A	Z
1011011	91	5B	[
1011100	92	5C	
1011101	93	5D	]
1011110	94	5E	^
1011111	95	5F	<
1100000	96	60	'
1100001	97	61	a
1100010	98	62	b
1100011	99	63	c
1100100	100	64	d
1100101	101	65	e
1100110	102	66	f
1100111	103	67	g
1101000	104	68	h
1101001	105	69	i
1101010	106	6A	j
1101011	107	6B	k
1101100	108	6C	l
1101101	109	6D	m
1101110	110	6E	n
1101111	111	6F	o
1110000	112	70	p
1110001	113	71	q
1110010	114	72	r
1110011	115	73	s
1110100	116	74	t
1110101	117	75	u
1110110	118	76	v
1110111	119	77	w
1111000	120	78	x
1111001	121	79	y
1111010	122	7A	z
1111011	123	7B	{
1111100	124	7C	|
1111101	125	7D	}
1111110	126	7E	~
1111111	127	7F	DEL

付録B
CP／M-86のファイル型

CP／M-86は，固有のファイル記述により各ファイルを識別する．ファイル記述はドライブ記述子，ファイル名それにファイル型から構成されている．ファイル型は省略可能な3文字のコードであり，ファイル名とピリオドによって区切られている．ファイル型は一般に特殊なファイルを示す．下記の表は，一般的なファイル型とその意味を示したものである．

表B-1.ファイル型

ファイル型	意味
A86	アセンブリ言語ソース・ファイル．CP／M-86のアセンブラであるASM-86はファイル型が．A86というファイルを機械言語にアセンブル，つまり変換する．
BAK	テキスト・エディタが作成するバック・アップ・ファイル．エディタはソース・ファイルをこのファイル型に変換し，オリジナル・ファイルが処理されたことを示す．オリジナル・ファイルはディスク上にバック・アップ・ファイルとして存在するので，そのファイルを参照することができる．
BAS	CBASIC／86プログラムのソース・ファイル
CMD	実行不可能な機械コードで命令を格納しているコマンド・ファイル
COM	8080の実行可能なファイル
H86	16進形式のプログラム・ファイル
INT	CBASIC／86プログラムの中間言語ファイル
LST	コンソールまたはプリンタに出力できる印刷可能なファイル
PRN	コンソールまたはプリンタに出力できる印刷可能なファイル
SUB	1つまたは複数のCP／M-86のコマンドを含む． SUBMIT入力ファイルのファイル型．SUBMITプログラムはファイル型がSUBであるファイルに格納されているコマンドを実行することにより，CP／M-86のバッチ・モードを提供する．
SYM	シンボル・テーブル・ファイル
$$$	PIPが作成する一時ファイル

付録C
CP／M-86の制御文字

表C-1. CP／M-86の制御文字

キーストローク	動作
BACKSPACE	カーソルを1文字分だけ後へ戻し，前の文字を消去する．
CTRL-C	指定されたコンソールで現在実行されているプログラムを中断する．
DEL	RUBと同じ．
CTRL-E	物理的なキャリッジ・リターンを行なうが，コマンドはCP／M-86に送られない．
CTRL-H	BACKSPACEと同じ．
CTRL-J	ライン・フィード，コンソールからの入力を終了する．
CTRL-M	キャリッジ・リターンと同じ．
CTRL-P	すべてのコンソール機能をプリンタにエコーする．もう1度CTRL-Pを入力するとプリンタへのエコーは停止する．この特殊文字が機能するのは，システムにプリンタが接続されている場合だけである．
CTRL-R	現在のコマンド行をタイプし直す．RUBまたはDELキーを使った後で役に立つ．
RETURN	キャリッジ・リターン
RUB	カーソルの左側にある文字を削除する．削除された文字はエコーされ，カーソルは右に動く．
CTRL-S	コンソールへのリストの出力を一時的に停止する．もう1回CTRL-Sを入力すると，リストの出力が再開される．
CTRL-U	行を取消し，#を表示する．カーソルは1行分だけ下へ移動し，新しいコマンドを待つ．
CTRL-X	コマンド行の中に含まれるすべての文字を削除する．
CTRL-Z	ストリングまたはフィールド・セパレータ．

付録C
CP/M-86の制御文字

[illegible]M-86の制御文字

動作	
カーソルを1文字分左に移動し、前の文字を削除する。	[illegible]
指定されたコンソール[illegible]	
RUBと同じ。	
[illegible]	
BACKSPACEと同じ。	
[illegible]を終了する。	
キャリッジ・リターンと同じ。	
[illegible]	
[illegible]	
キャリッジ・リターン	
[illegible]	RUB
[illegible]	CTRL-S
[illegible]	CTRL-U
[illegible]	CTRL-X
[illegible]	CTRL-Z

付録D
CP／M-86のエラー・メッセージ

CP／M-86コマンド・メッセージ

メッセージ　　　意　味

```
Ambiguous operand
```

DDT-86．あいまいなオペランドを持つコマンドをアセンブルしようとすると，このメッセージが出力される．この場合には，先行文字“BYTE”または“WORD”を持つオペランドを指定しなければならない．

```
Bad Directory on d:
Space Allocation Conflict:
User n  d:filename.typ
```

データ・ブロックが複数のファイルに割当てられる場合，領域割当てに矛盾のあることをSTATが検出すると，このメッセージが出力される。この場合，1つまたは複数のファイル名が表示される．示される各ファイルには，ディスク上の別のファイルに既に割振されているデータ・ブロックが含まれる．この問題を修正するには，表示されたファイルを消去しなければならない．矛盾のあるファイル（1つまたは複数）を消去した後，↑Cを押すことにより割振りベクトルを再生成する．この処理を行なわなければ，同じエラーが何回も繰返される．

```
BDOS err on d:
```

CP／M-86はd：の代わりにエラーの発生したドライブのドライブ記述子を使用する．このメッセージが出力されるのは，CP／M-86がドライブの中にディスクがセットされていないことを検出したとき，ディスクが正しくフォーマッティングされていない，ドライブ・ラッチが開いている，あるいはドライブの電源が入っていない場合である．したがって，これらの状況が発生していないかどうかを確認し，もう一度コマンドを実行する．

```
BDOS err on d:  bad sector
```

このメッセージはハードウェアに問題があるか，あるいはディスクが正しくフォーマッティングされていないことを示す．この場合には，CTRL-Cを入力し，プログラムを終了し，CP／M-86にもどり，エラーを無視するようにENTERキーを押す．

```
BDOS err on d:  select
```

CP／M-86が受取った要求が，存在しないドライブを指定したものであったり，ドライブ中のディスクが正しくフォーマッティングされていない場合には，このメッセージが出力される．CP／M-86は，ユーザーが任意のキーを押すと，直ちに現在のプログラムを終了する．

```
BDOS err on d:  RO
```

ドライブがSTATコマンドによって読取り専用ステータスに割当てられている，あるいはドライブの中のディスクが交換された後，CTRL-Cによって初期化されていない場合に，このメッセージが出力される。CP／M-86はユーザが任意のキーを押すと，直ちに現在のプログラムを終了する．

```
Cannot close
```

ASM-86．出力ファイルをクローズできないと，このメッセージが出力される．これは回復不可能なエラーであり，ASM-86の実行は終了する．この場合，ユーザは正しいディスクがドライブにセットされているかどうか，またそのディスクが書込み保護されていないかどうかを確認した後で，適切な処理を取らなければならない．

DDT-86．Wコマンドによって書込まれたディスク・ファイルをクローズできない場合にも，このメッセージが出力される．これは回復不可能なエラーであり，DDT-86の実行は終了する．この場合，ユーザは正しいディスクがドライブにセットされているかどうか，またディスクが書込み保護されていないかどうかを確認した後，適切な処置を取らなければならない．

Command name?

CP／M-86がユーザの指定したコマンドを見つけることができない場合には，入力したコマンド名の後に疑問符（?）を付加したメッセージを出力する．この場合，ユーザはコマンド名を正しくタイプしたかどうか，また要求したコマンドが，CMDファイルとして省略時ディスクまたは指定したディスクに存在するかどうかを確認しなければならない。

DESTINATION IS R/O, DELETE (Y/N)?

PIP．PIPコマンドに指定されている目的ファイルが既に存在し，それが読取り専用である場合，このメッセージが出力される．ユーザがYをタイプすると，ファイル・コピーが行なわれる前に目的ファイルが削除される．

Directory full

ASM-86。出力ファイルのためのディレクトリ領域が十分に存在しない場合，このメッセージが出力される．この場合には，不必要なファイルを消去するか，より多くのディレクトリ領域を持つ別のディスクをセットし，ASM-86を再び実行する．

Disk full

ASM-86．出力ファイル（LST，H86，SYM）を書込むための十分なディスク領域がないとき，このメッセージが出力される．この場合には，不必要なファイルを消去するか，多くの領域を含む別のディスクをセットし，ASM-86を再び実行する．

Disk read error

ASM-86．ソース・ファイルまたは中間ファイルを正しく読み取ることができない場合，このメッセージが出力される．このメッセージが出力されるのは，通常，期待されていないときにファイルの終りに到達した場合である．この場合には，問題点をソース・ファイルの中で修正しなければならない．

DDT-86．Rコマンドに指定されているディスク・ファイルを正しく読取ることができない場合，このメッセージが出力される．このエラーが発生するのは，通常期待されていないときに，ファイルの終りに到達した場合である．この場合は，問題点をファイルの中で修正しなければならない．

Disk write error

DDT-86．Wコマンドを実行している間におそらくディスクが一杯になったためにディスクへの書込みオペレーションを正しく実行できないとき，このメッセージが出力される．この場合には，不必要なファイルを消去するか，多くの領域を持つ別のディスクをセットし，ASM-86を再び実行する．

Double defined variable

ASM-86．変数の名前として使用されている識別子が，変数またはラベルの名前としてプログラムの他の場所で使用していると，このメッセージが出力される．

```
例：X    DB    5
         ・・・
    X    DB    123H
```

Double defined label

ASM-86．ラベルとして使用されている識別子が，ラベルまたは変数子としてプログラムの他の場所で使用していると，このメッセージが出力される．

```
例：LAB3:  MOV    BX,5
           ・・・
    LAB3:  CALL   MOVE
```

Double defined symbol - treated as undefined

ASM-86. EQUディレクティブの名前として使用されている識別子が，別の名前としてプログラムの他の場所で使用していると，このメッセージが出力される．

ERROR: BAD PARAMETER

PIP. PIPコマンドに誤ったパラメータが入力されると，このメッセージが出力される．この場合には，パラメータを正しくタイプし直す．

ERROR: CLOSE FILE - {filespec}

PIP. 出力ファイルをクローズできないと，このメッセージが出力される．この場合，ユーザは正しいディスクがドライブにセットされているかどうか，またディスクが書込み保護されていないかどうかを確認した後，適切な処置を取らなければならない．

ERROR: DISK READ - {filespec}

PIP. PIPコマンドに指定されている入力ディスク・ファイルを正しく読取ることができないとき，このメッセージが出力される．このエラーが発生するのは，通常，期待されないときにファイルの終りに到達した場合である．この場合には，ファイルの中で問題点を修正しなければならない．

ERROR: DISK WRITE - {filespec}

PIP. PIPコマンドを実行している間に，おそらくディスクが一杯になったためにディスクへの書込みオペレーションを実行できないとき，このメッセージが出力される．この場合には，不必要なファイルを消去するか，多くの領域を持つ別のディスクをセットし，PIPを再び実行する．

ERROR: FILE NOT FOUND - {filespec}

PIP. ユーザの指定した入力ファイルが存在しない場合，このメッセージが出力される．

ERROR: HEX RECORD CHECKSUM - {filespec}

PIP. 16進ファイルを転送しているときに，16進数のレコード・チェックサムが検出されると，このメッセージが出力される．この場合には，チェックサムの誤っている16進ファイルを修正しなければならないが，おそらくその16進ファイルを作成し直す必要があるのであろう．

Error in codemacro building

ASM-86. コードマクロの中に誤まったステートメントが含まれているか，コードマクロ・ディレクティブがコードマクロの外部で検出された場合，このメッセージが出力される．

ERROR: INVALID DESTINATION

PIP. PIPコマンドの中に指定されている宛先が誤まっている場合，このメッセージが出力される．このようなエラーが発生するのは，おそらく宛先として入力装置を指定した場合である．

ERROR: INVALID FORMAT

PIP. PIPコマンドの形式が誤まっていると，このメッセージが出力される．この場合には，PIPコマンドの説明を参照されたい．

ERROR: INVALID HEX DIGIT - {filespec}

PIP. 16進ファイルを読取っているときに，誤まった16進数が検出されると，このメッセージが出力される．この場合には，誤まった16進数を含む16進ファイルを修正しなければならないが，おそらくその16進ファイルを作成し直さなければならないだろう．

ERROR: INVALID SEPARATOR

PIP. 2つの入力ファイル名の間のセパレータとして誤った文字を指定した場合，このメッセージが出力される.

ERROR: INVALID SOURCE

PIP. PIPコマンドに指定されているソースが誤っている場合，このメッセージが出力される．このようなエラーが発生するのは，おそらくソースとして出力装置を指定した場合でもある.

ERROR: INVALID USER NUMBER

PIP. 15より大きいユーザ番号を指定すると，このメッセージが出力される．ユーザ番号は0から15の範囲でなければならない.

ERROR: NO DIRECTORY SPACE - {filespec}

PIP. 出力ファイルのためのディレクトリ領域が十分でないと，このメッセージが出力される．この場合には，不必要なファイルを消去するか，多くのディレクトリ領域を含む別のディスクをセットし，PIPを再び実行する.

ERROR: QUIT NOT FOUND

PIP. Qパラメータに対するストリング引数が入力ファイルの中に指定されていない場合，このメッセージが出力される.

ERROR: START NOT FOUND

PIP. Sパラメータに対するストリング引数がソース・ファイルの中に指定されていない場合，このメッセージが出力される.

ERROR: UNEXPECTED END OF HEX FILE - {filespec}

PIP. 最後の16進レコードの前にファイルの終りが検出されると，このメッセージが出力される．この場合，終了レコードのない16進ファイルは修正しなければならないが，おそらくその16進ファイルを作成し直さなければならないだろう.

ERROR: USER ABORTED

PIP. ユーザがあるキーを押すことによってPIPオペレーションを途中で打切った場合，このメッセージが出力される.

ERROR: VERIFY - {filespec}

PIP. Vオプションを使ってコピーを行なう場合，PIPは書込んだデータを再び読取り，それをメモリ・バッファのデータと比較する．このときに両方のデータに相違点があると，このメッセージが出力される．このようなエラーが発生するのは，通常，目的ディスクまたは目的ドライブに障害がある場合である.

File exists

既に別のファイルを割当てられているファイルの記述を使って新しいファイルを作成するようにCP/M-86に指示すると，このエラー・メッセージが出力される．この場合には，既存のファイルを削除するか，別のファイル記述を使用しなければならない.

File name syntax error

ASM-86. INCLUDEディレクティブの中に指定されているファイル名が誤った形式であると，このメッセージが出力される.

例：INCLUDE FILE.A86X

File not found

CP／M-86が指定されたファイルを見つけることができない場合，このメッセージが出力される．このとき，ユーザは正しいドライブ記述が入力されているかどうか，また正しいディスクがドライブにセットされているかどうかを確認しなければならない．

Garbage at end of line - ignored

ASM-86．ASM-86が行の終りを期待しているときに，その行からさらに別の項目が検出されると，このメッセージが出力される．

例：
```
NOLIST 4
MOV     AX,4     RET
```

Illegal expression element

ASM-86．表現式の形式が誤っている場合，このメッセージが出力される．

例：
```
X       DB      12X
        DW      (4 * )
```

Illegal first item

ASM-86．ソース行の最初の項目が正しい識別子，ディレクティブまたはニーモニックでない場合，このメッセージが出力される．

例：
```
1234H
```

Illegal "IF" operand - "IF" ignored

ASM-86．IFステートメントの中の表現式が数値でないか，あるいはその中に前方向参照が含まれている場合，このメッセージが出力される．

Illegal pseudo instruction

ASM-86．擬似命令の前に必要な識別子が脱落しているか，識別子を指定することのできない擬似命令の前に識別子が指定されている場合，このメッセージが出力される．

Illegal pseudo operand

ASM-86．ディレクティブの中のオペランドが誤っている場合，このメッセージが出力される．

例：
```
X       EQU     0AGH

        TITLE   UNQUOTED STRING
```

Instruction not in code segment

ASM-86．命令CSEG以外のセグメントに指定されていると，このメッセージが出力される．

Is this what you want to do (Y/N)?

COPYDISK．表示されているCOPYDISKと機能がユーザの希望するものなら，Yをタイプする．

Insufficient memory

DDT-86．RまたはEコマンドに指定されているファイルをロードするのに十分なメモリがない場合，このメッセージが出力される．

Invalid Assignment

STAT. STAT装置割当てに誤った装置が指定されている場合，このメッセージが出力される．4つのSTAT論理装置であるCON:，RDR:，PUN:，LST:のそれぞれに対する正しい割当てをリストするにはSTAT val:を使用する．

Label out of range

ASM-86. コール，ジャンプまたはループ命令の中で参照されているラベルが範囲の外側にあると，このメッセージが出力される．ラベルはその命令を含むセグメント以外のセグメントで定義できる．しかし，ショート命令の場合（JMPS，条件つきジャンプおよびループ)，ラベルが次の命令の記憶位置から128バイト以上離れているとエラーとなる．

Memory request denied

DDT-86. Rコマンドを実行しているときに，メモリに対する要求が満足されないと，このエラー・メッセージが出力される．1度に割振ることのできるメモリのブロック数は最大ブロックである．

Missing instruction

ASM-86. ソース行の接頭語の後に命令が続かない場合，このメッセージが出力される．

例： REPNZ

Missing pseudo instruction

ASM-86. ソース行の最初の項目が正しい識別子であり，2番目の項目がその識別子の後に指定できる正しいディレクティブでないときに，このエラー・メッセージが出力される．

例：THIS IS A MISTAKE

Missing segment information in operand

ASM-86. CALLFまたはJMPF命令のオペランド（またはDDディレクティブの表現式）の中にセグメント情報が含まれていない場合，このメッセージが出力される．必要なセグメント情報は，下記のようにセグメント・ディレクティブに数値フィールドを指定することによって与えることができる．

```
        CSEG    1000H
X:
        . . .
        JMPF    X
        DD      X
```

Missing type information in operand(s)

ASM-86. 命令オペランドの中に十分なタイプ情報が含まれていない場合，このメッセージが出力される．

例：MOV   [BX],10

Nested "IF" illegal - "IF" ignored

ASM-86. IFステートメントに対する最大入力子レベルを越えた場合，このメッセージが出力される．

Nested INCLUDE not allowed

ASM-86. INCLUDEディレクティブが既に含まれているファイルを検出した場合，このメッセージが出力される．

No file

CP/M-86が指定されたファイルを見つけることができない，またはそのようなファイルが存在しない場合，このメッセージが出力される．

ASM-86．指定されたソース・ファイルまたはインクルード・ファイルが，指定されたドライブから見つけることができない場合，このメッセージが出力される．

DDT-86．RまたはEコマンドに指定されているファイルをディスクから見つけることができない場合，このメッセージが出力される．

No matching "IF" for "ENDIF"

ASM-86．ENDIFステートメントだけが検出され，対応するIFステートメントが存在しないと，このメッセージが出力される．

No space

DDT-86．Wコマンドによって書込まれるファイルのためのディスクトリ領域が存在しない場合，このメッセージが出力される．

Operand(s) mismatch instruction

ASM-86．命令の中に含まれているオペランドの数が誤っているか，あるいはオペランドのタイプが一致しない場合，このメッセージが出力される．

例：

```
        MOV     CX,1,2
X       DB      0
        MOV     AX,X
```

Parameter error

ASM-86．ASM-86コマンドのコマンド・テイルの中に含まれるパラメータが誤って指定されている場合，このメッセージが出力される．

例：`ASM86 TEST $S;`

Symbol illegally forward referenced - neglected

ASM-86．ORG，RS，EQUまたはIFステートメントの中で指定されているシンボルが誤って前方向参照されている場合，このメッセージが出力される．

Symbol table overflow

AMS-86．シンボル・テーブルのためのメモリ領域が十分でないと，このメッセージが出力される．この場合には，シンボルの長さ，または数を少なくするか，使用可能なメモリ領域を増加してアセンブルをしなおさなければならない．

Undefined element of expression

ASM-86．オペランドとして使用されている識別子が定義されていない，または誤って前方向参照されている場合，このメッセージが出力される．

例：

```
        JMP     X
A       EQU     B
B       EQU     5
        MOV     AL,B
```

Undefined instruction

ＡＳＭ-86．ソース行のラベルの後の項目が正しい命令でない場合，このメッセージが出力される．

例：DONE:　BAD　INSTR

Use: [size] [ro] [rw] [sys] or [dir]

ＳＴＡＴ．このメッセージが出力されるのは，ファイル属性をセットするコマンドが誤っているときである．ＳＴＡＴ　ファイル記述〔オプション〕コマンドで指定できるオプションは，ここに示されているものだけである．

Use:　STAT d:=ROM

ＳＴＡＴ．誤ったＳＴＡＴドライブ・コマンドが与えられると，このメッセージが出力される．ＳＴＡＴでドライブとして指定できるのは，ＳＴＡＴ　ｄ　＝ＲＯだけである．

Too Many Files

ＳＴＡＴ．万能札を含むＳＴＡＴコマンドが，ＳＴＡＴで分類できる以上のファイルと一致したとき，このメッセージが出力される．ＳＴＡＴは最大512のファイルを分類することができる．

Verify error at s:o

ＤＤＴ-86．充塡，セット，転送，またはアセンブル・コマンドによってメモリに記憶された値を正しく読取ることができない場合，このメッセージが出力される．このようなエラーが発生するのは，ユーザ・メモリの指定が誤っているか，ＲＯＭまたは存在しないメモリにデータを書込もうとした場合である．

付録E
用語集

あいまいなファイル名（ambiguous filename）

基本ファイル名とファイル型のどちらか一方，または両方にＣＰ／M-86のワイルド・カード文字である？または* を含むファイル名．ファイル名の中の文字をこれらのワイルド・カード文字と置き換えれば，そのファイル名はあいまいなファイル名となり，１つのコマンド行で複数のＣＰ／M-86のファイルを容易に参照できるようになる．本章の第２章を参照．

アプリケーション・プログラム（applications program）

実行するために環境をオペレーティング・システムが提供しなければならないプログラム．代表的なアプリケーション・プログラムとしては事務処理用会計パッケージ，ワード・プロセシング（編集）・プログラムおよび郵便リスト・プログラムがある．

引数（argument）

問題の公式に適切な意味を与えるために数字，文字，あるいは名前を使用することのできる位置を示す記号で，一般に英字である．

ＡＳＣＩＩ：情報交換用米国標準コード（American Standard Code for Information Interchange）

これは，数字，英字，記号を表現するための標準的なコードである．ＡＳＣＩＩテキスト・ファイルはスクリーンに表示したり，用紙に印刷したりできるファイルである．付録Ａを参照．

属性（attribute）

オン，またはオフにセットできるファイル特性．

バック・アップ（back-up）

ディスクまたはファイルを安全に保存するために作成される，ディスクまたはファイルのコピー．

ビット（bit）

オン（１）またはオフ（０）にセットできるメモリの中の〝Switch〞．ビットはバイトにグループ化される．

ブロック（block）：

特定の使い方をするために確保されているディスク領域．

ブートストラップ（bootstrap）

オペレーティング・システムをメモリにロードする処理．ブートストラップ手順はシステムによって異なる．それぞれのオペレーティング・システムのブートは，そのオペレーティング・システムが管理するメモリ・サイズおよびハードウェア環境に合わせてカストマイズしなければならない．電源を入力したとき，あるいはコンピュータがリセットされた場合には，一般に自動的にブートがロードされ，実行される．このような処理を〝コールド・スタート〞と呼ぶことがある．

バッファ（buffer）：

情報を転送するときに，一時的にデータを記憶するメモリ領域．

組込みコマンド（built-in commands）：

メモリに常駐するコマンド．これからのコマンドはディスクからアクセスされるわけではないので，直ちに応答する．

バイト（byte）：

８ビットより構成されるメモリまたはディスク領域の単位．

コマンド (command)：

ＣＰ／M-86コマンド行のエレメント．一般にＣＰ／M-86のコマンドは３つの部分に分かれている．それはコマンド・キーワード，コマンド・テイル，そしてキャリッジ・リターンである．

コマンド・ファイル (command file)：

ディスクにプログラム・ファイルとして格納されているコード化された機械で実行可能な一連の命令であり，コンソールにシステム・プロントが表示された後，コマンド・キーワードを入力することにより，ＣＰ／M-86が呼出すことができるものである．ＣＰ／M-86のコマンド・ファイルのファイル型は，一般にＣＭＤである．ファイルはコマンド・ファイルかデータ・ファイルのいずれかである．コマンド・プログラムも同じである．

コマンド・キーワード (command keyword)：

ＣＰ／M-86のコマンド。一般にファイル型がＣＭＤのファイルの基本ファイル名，または組込みコマンドを識別する名前．コマンド・キーワードは，コマンド行のコマンド・テイルおよびキャリッジ・リターンの前に指定される．

コマンド構文 (command syntax)：

コマンドを入力するための正しい方法を定義するステートメント．コマンドの正しい構造には，一般にコマンド・キーワード，コマンド・テイル，それにキャリッジ・リターンが含まれる．構文行には一般に，コマンドを入力するときに実際の値と置換えなければならない記号が含まれている．

コマンド・テイル (command tail)：

コマンド行のコマンド・キーワードの後に指定されるコマンドの一部で，コマンド・テイルにはドライブ語述，ファイル名，ファイル型，それにオプションやパラメータを指定することができる．コマンドの中にはコマンド・テイルを必要としないものもある．

連結 (concatenate)：

複数の独立したファイルを，指定されている順に１つの新しいファイルにコピーするＰＩＰオペレーションの１つを説明する用語．

コンソール (console)：

基本入出力装置．コンソールはスクリーンなどのリスト装置とキーボードから構成されており，ユーザはコンソールを通じてオペレーティング・システムやアプリケーション・プログラムと通信する．

制御文字 (control character)：

簡単なコマンドをＣＰ／M-86に送るための印刷できない文字の組合せ，制御文字の中には行編集機能を実行するものもある．制御文字を入力するには，端末装置のＣＯＮＴＲＯＬキーを押したままで，指定されている文字キーを押す．付録Ｃを参照．

カーソル (cursor)：

スクリーンに表示することのできる１文字の記号．カーソルはコンソールから次に入力されるデータがどこにセットされるかを示す．

データ・ファイル (data file)：

一般には，処理するためにコマンド・ファイルを必要とする類似した実行不可能な情報の集まり．

省略値 (default)：

現在，選択されているディスク・ドライブおよびユーザ番号．ディスク・ドライブやユーザ番号を指定していないコマンドは省略時ディスク・ドライブおよびユーザ番号を参照する．ＣＰ／M-86が最初に呼出されるとき，省略時ディスク・ドライブはドライブＡであり，ユーザ番号は０である．この状態はＵＳＥＲコマンドによって特に変更されない限り有効である．

区切り文字（delimiter）：
コマンド行の中の異なる項目を区切るために使用される特殊な文字．たとえば，ＣＰ／Ｍ-86では，コロンはドライブ記述とファイル名とを区切る．ピリオドはファイル名とファイル型を区切る。角括弧はオプションとコマンドまたはファイル記述を区切る．カンマはオプション・リストの中の各項目を区切る．上記の特殊文字はすべて区切り文字である．

ディレクトリ（directory）：
ディスク上の各ファイルに対するエントリを含むディスクの一部．ＤＩＲコマンドに対する応答として，ＣＰ／Ｍ-86はディレクトリに記憶されているファイルのファイル名を表示する．

ＤＩＲ属性（ＤＩＲ attribute）：
ファイル属性．ＤＩＲ属性を持つファイルは，ＤＩＲコマンドを使って表示することができる．このファイルは省略時ユーザ番号および省略時ドライブからだけしかアクセスできない．

ディスク，ディスケット（disk, diskette）：
情報を記憶するために使用される磁気媒体．プログラムおよびデータは，音楽がカセット・テープに記録されるのと同じ方法でディスクに記録される．〝diskette〞という用語は記憶容量の小さい着脱可能なフロッピー・ディスケットを示す．〝disk〞という用語はディスケット，着脱可能なカートリッジ・ディスク，または固定ハード・ディスクを示す．

ディスク・ドライブ（disk drive）：
ハード・ディスク，またはフロッピー・ディスクにデータを書込んだり，これらのディスクからデータを読取ったりする周辺装置．ＣＰ／Ｍ-86はその制御のもとで各ドライブに１文字を割当てる．たとえば，ＣＰ／Ｍ-86は４ドライブ・システムのドライブをＡ，Ｂ，Ｃ，Ｄとして参照する．

エディタ（editor）：
テキスト・ファイルを作成し，変更するユーティリティ・プログラム．エディタはコンピュータ・プログラムのためのコードを作成したり，文章を作成するために使用することができる．
ＣＰ／Ｍ-86のエディタは，コンソールにシステム・プロンプトが表示された後でコマンドＥＤをタイプすることにより呼出される．（本書の第５章のＥＤを参照）．

実行可能（executable）：
コンピュータが実行できる状態．実行可能なコードとは，コンピュータが実行できる一連の命令である．たとえば，コンピュータは名前や住所を〝実行〞できないが，郵送用ラベルにそのような名前や住所をすべて印刷するプログラムを実行することは可能である．

プログラムの実行（execute a program）：
プログラムの実行を開始する．プログラムが実行されている間，コンピュータは一連の命令を実行する．

ＦＣＢ：
ファイル制御ブロック

ファイル（file）：
ディスクに記憶されている文字，命令またはデータの集まり．ユーザはディスクにファイルを作成することができる．

ファイル制御ブロック（File Control Block）：
ディスク上のファイルをアクセスするために用いられる構造．ドライブ，ファイル名，ファイル型およびディスクからアクセス，または生成したいファイルを記述するその他の情報が含まれる．

ファイル名（file name）：

ファイルに割当てられた名前．ファイル名には1文字から8文字の基本ファイル名と0文字から3文字のファイル型とが含まれる．ピリオドは基本ファイル名とファイル型を区切る．

ファイル記述（file specification）

固有のファイル識別子．完全なCP／M-86のファイル記述には,ディスク・ドライブ記述とコロン（d：），1文字から8文字の基本ファイル名，ピリオド，それに0文字から3文字のファイル型が含まれる．たとえば，b：example. texは完全なCP／M-86のファイル記述である．

ファイル型（filetype）

ファイル名の拡張．ファイル型は0文字から3文字であり，基本ファイル名とピリオドで区切らなければならない．ファイル型はファイルに関する特定の情報を提供する．ファイルを処理する必要のある特定のプログラムには特定のファイル型が割当てられる（付録Bを参照）．

フロッピー・ディスク（floppy disk）

情報を記憶するために使用されるフレキシブル磁気ディスク．フロッピー・ディスクには直径が5¼インチのものと，8インチのものがある．

ハード・ディスク（hard. disk）

コンテナに封入されている固定したレコード盤のような磁気ディスク．ハード・ディスクにはフロッピー・ディスクより多くの情報を記憶することができる．

ハードウェア（hardware）

コンピュータの物理的なコンポーネント．

16進ファイル（hex file）

コマンド（機械用語）ファイルをＡＳＣIIの印刷可能な表現に直したもの．

16進表記（hexadecimal notation）

16進数のための表記法で，16の数字を表現するために0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F．という記号を使用する．16進表記は容易にＡＳＣII文字によって表現することができ，コンソールのスクリーンやプリンタに出力することができるので，機械コードは16進表記に変換されることが多い（付録Aを参照）．

入力（input）

コンピュータに入るデータ．一般に端末装置からオペレータがタイプするか，プログラムがディスクから読取る．

インタフェース（interface）

マイクロコンピュータにおけるハードウェアとソフトウェアの間のインタフェースと同様に，2つの独立したシステムを互いに通信できるようにするもの．

I／O

input／output（入出力）の短縮型．

キーワード（keyword）

コマンド・キーワードを参照．

キロバイト（kilobyte）

1024バイトは1kと表わされる．32キロバイトは32kである．1024キロバイトは1メガバイトであり，これは100万バイトより大きい数字である．

リスト装置（list device）

プリンタなどのような装置で，データをリスト，またはプリントすることができる．

ログイン（logged in）

ドライブをオペレーティング・システムに登録すること．ユーザまたは実行中のプロセスがドライブを選択すると，そのドライブはログインされ，ユーザがフロッピー・ディスク・ドライブのディスクを交換するか，コマンドレベルで↑Cを入力するまで，そのドライブは選択されたまま，つまりログインされたままの状態である．

論理（logical）

実際の物理的な形式と同じであるか，または同じではない表現．たとえばハード・ディスクは1つの物理ドライブを占有するが，ユーザはそのドライブの記憶領域を分割することにより，複数の異なるドライブがあるかのように考えることができる．これらの独立したドライブが論理ドライブである．

メガバイト（megabyte）

100バイトを上まわるバイト数であり，1024キロバイトである（バイト，キロバイトの項を参照）

マイクロプロセッサ（microprocesser）

マイクロコンピュータの中央演算処理装置（ＣＰＵ）となるシリコン・チップ．

オペレーティング・システム（operating system）

他のプログラムの実行を監視し，コンピュータ資源を管理するプログラムの集まり．オペレーティング・システムはコンピュータと周辺装置の間の秩序正しい入出力環境を提供する．オペレーティング・システムを使用することにより，ユーザの作成したプログラムは完全に実行される．

オプション（option）

コマンド・テイルの一部を構成する多くのパラメータの1つ．オプションを使用すれば，コマンドの実行に対して特定の条件を指定することができる．

出力（output）

プロセッサがコンソールまたはディスクに送るデータ．

パラメータ（parameter）

コマンドに追加情報を提供するコマンド・テイルの中の値．技術的には，パラメータはコマンドに必ず必要なエレメントである．

周辺装置（peripheral device）

ＣＰＵの外部の装置．たとえば端末装置，プリンタ，ディスク・ドライブは一般的な周辺装置であり，これらはプロセッサの一部ではないが，プロセッサと一緒に使用される．

物理（physical）

コンピュータの実際のハードウエア．物理的な環境はコンピュータによって異なる．

基本ファイル名（primary filename）

ファイル名の最初の8文字．基本ファイル名は固有名であり，ユーザがファイルの内容を識別するのに役立つ．基本ファイル名は1文字から8文字で構成され，英字や数字，それに特殊文字を含むことができる．基本ファイル名は省略可能なドライブ記述とファイル型の間に指定する．

プログラム (program)

コンピュータで実行するときに，特定の仕事を実行するようにコード化された命令の集まり．

プロンプト (prompt)

スクリーンに表示される文字で，次にどのような動作が必要かをユーザが判断するのに役立つ．システム・プロムプトはオペレーティング・システムが表示する特別なプロムプトである．システム・プロンプトは，オペレーティング・システムが入力を受取れる状態であることをユーザに示す．ＣＰ／M-86のシステム・プロンプトは英字の後に＞が続いたものである．英字は省略時ドライブを示す．アプリケーション・プログラムの中には，独自の特殊な〝システム〞プロンプトを持つものがある．

読取り専用 (Read-Only)

ディスク・ファイルおよびディスク・ドライブに割当てることのできる属性．ファイルに読取り専用属性を割当てると，ユーザはそのファイルからデータを読取ることはできるが，内容を変更することはできない．ドライブに読取り専用属性を割当てると，ユーザはディスク上のファイルを読取ることはできるが，新しいファイルを追加したり，ファイルを消去したり，変更したり，ファイル名を変更したり，あるいはディスクに書込みを行ったりすることはできない．ＳＴＡＴコマンドは，ファイルおよびドライブを読取り専用属性にセットすることができる．ファイルやドライブは読取り専用か読み書きステータスのいずれかである．ドライブおよびファイルに対する省略時セッティングは読み書きステータスであるが，ディスクをリセットしたり，媒体を自動的に交換するときにエラーが発生すると，そのエラーが修正されるまでドライブは読取り専用にセットされる．ファイルおよびディスク・ドライブは読取り専用ステータスにも読書きステータスにもセットすることができる．

読み書き (Read-Write)

ディスク・ファイルおよびディスク・ドライブに割当てることのできる属性．読み書き属性を割当てると，特定の読み書きファイル，または読み書き状態にセットされているドライブのディスク上に存在するファイルを読取ったり，書込んだりすることができる．ファイルまたはドライブは読取り専用ステークス，または読み書きステークスにセットすることができる．

レコード (record)

データの集まり．ファイルはディスクに記憶されている１つまたは複数のディスクから構成される．

ＣＰ／M-86のレコードの長さは128バイトである．

RO

読取り専用 (Read-Only) の短縮形．

RW

読み書き (Read-Write) の短縮形．

セクタ (Sector)

ディスク・トラックの一部．各トラックには指定された数のセクタが存在する．

ソフトウェア (software)

機械で読取りできる命令をコンピュータに送るために，特別にコード化されたプログラム．ハードウェアに対する用語であり，ハードウェアはコンピュータの実際の物理的なコンポーネントである．

ソース・ファイル (source file)

エディタ，テキスト・フォーマットあるいはアセンブラなどの，処理プログラムのための入力ファイルであるＡＳＣⅡテキスト・ファイル．

構文 (syntax)

与えられたコマンドを入力するための形式．

システム属性（system attribute）

ファイル属性．システム属性をファイルに割当てるにはＳＴＡＴコマンドでＳＹＳオプションを使用する．ＳＹＳ属性を持つファイルはＤＩＲコマンドに対する応答で表示されない．したがって，ＳＹＳ属性を持つファイルを表示するにはＤＩＲＳを使用しなければならない（第4章を参照）．ユーザ番号から読取り，実行できるようになる．この機能を利用すれば，一般に使用されるプログラムをどのユーザ番号からでもアクセスできるようになる．

システム・プロンプト（system prompt）

システムが入力を受取れる状態であることを示すために，オペレーティング・システムが表示する記号．プロンプトの項を参照．

端末（terminal）

コンソールの項を参照．

トラック（track）

ディスクを分割する同心円．一般的な8インチのフロッピー・ディスクには77本のトラックが存在する．

ターンキー・アプリケーション（turn-key application）

コンピュータの専門家ではないユーザのために設計されたアプリケーション．たとえば，代表的なターンキー・アプリケーションは，オペレータがコンピュータの電源を入れ，正しいプログラム・ディスクを挿入し，スクリーンに表示されている機能（メニュー）の中から希望する手順を選択するだけでよいように設計されている．

上位互換性（upward-compatible）

既にリリースされているオペレーティング・システム（またはコンパイラ等）のために作成されているプログラムが，同じオペレーティング・システムの新しくリリースされたバージョンのもとで実行できることを示す用語．

ユーザ番号（user number）

ディスク・ディレクトリのファイルに割当てられる番号で，それぞれのユーザが自分のファイルだけを取扱えばよく，すべてのファイルが同じディスク上に存在する場合でも各ユーザは〝独自の〟ディレクトリを持つことができる．CP／M-86ではファイルを16のユーザ・グループに分割することができる．

ユーティリティ（utility）

〝ツール〟。ファイルのコピーやファイルの削除あるいはファイルの編集などの特定のオペレーションを，ユーザが実行できるようにするためのプログラム．ユーティリティはプログラマやユーザが便利なように作成されるものである．

ワイルド・カード文字（wildcard characters）

特定の項目と対応する特殊な文字．ＣＰ／M-86には2つワイルド・カード文字がある．それは？と*である．？はファイル名の中の任意の1文字と置換えることができ，*は基準ファイル名，またはファイル型あるいはその両方と置換えることができる．ファイル名にワイルド・カード文字を指定すれば，ユーザはあいまいなファイル名を作成することができ，1つまたは複数のファイルをすみやかに参照することができる．

CP/M-86
システムガイド

まえがき

CP／M-86システム・ガイドは，CP／M-86™のシステム・プログラミングの説明である．ここでCP／M-86™は，インテル8086と8088，16ビット・マイクロプロセッサに対する単一ユーザのためのオペレーティング・システムであり，ここでの論議は，読者がDigital Research 8ビット　オペレーティング・システムCP／Mをよく理解しているものと仮定している．

CP／M-86との相違を明らかにするために，このドキュメントでは，CP／Mの8ビット・バージョンはCP／M-80™と呼ばれており，両方のシステムに共通な要素は，単にCP／Mの特徴と呼ばれている．

このシステム・ガイドは，CP／M-86プログラミング・インタフェースの規約の概略を述べており，また顧客のハードウェア環境にCP／M-86を適合させるための手順をも説明している．この情報は，CP／M2.0インタフェース説明書とCP／M2.0改造説明書とに述べたことと対応している．

1章は，CP／M-86を概説し，CP／M-80との相違を要約して述べている．2章は，一般的な実行環境を説明している．3章は，コマンド・ファイルの作成方法を述べている．4章と5章では，それぞれ基本ディスク・オペレーティング・システムと基本入出力システムとのプログラミング上のインタフェースを定義している．6章は，顧客のディスク構成をサポートするためのBIOSの変更点を説明している．7章は，ローディング・オペレーションとCP／M-86システム・ファイルの構成とを述べている．

目　次

1. CP／M-86システムの概要 ……… 1
- 1.1 CP／M-86の一般的特徴 ……… 1
- 1.2 CP／M-80とCP／M-86との差異 ……… 3

2. CP／M-86のコマンド・セットアップと実行 ……… 5
- 2.1 CCPビルドイン・コマンドとトランジェント・コマンド ……… 5
- 2.2 トランジェント・プログラムの実行モデル ……… 6
- 2.3 8080メモリ・モデル ……… 7
- 2.4 スモール・メモリ・モデル ……… 8
- 2.5 コンパクト・メモリ・モデル ……… 9
- 2.6 ベース・ページの初期化 ……… 10
- 2.7 トランジェント・プログラムのロードと出口 ……… 11

3. コマンド (CMD) ファイルの生成 ……… 12
- 3.1 Intel 8086 Hexファイル形式 ……… 12
- 3.2 GENCMDのオペレーション ……… 13
- 3.3 LMCMDのオペレーション ……… 15
- 3.4 コマンド (CMD) ファイルの形式 ……… 15

4. 基本ディスク・オペレーティング・システムの機能 ……… 17
- 4.1 BDOSのパラメータと機能コード ……… 17
- 4.2 単純なBDOS機能の呼び出し ……… 19
- 4.3 BDOSのファイル操作 ……… 23
- 4.4 BDOSメモリ管理とロード ……… 37

5. 基本I／Oシステム (BIOS) の構成 ……… 41
- 5.1 BIOSの構成 ……… 41
- 5.2 BIOSジャンプ・ベクトル ……… 42
- 5.3 周辺装置 ……… 43
- 5.4 BIOSサブルーチン・エントリー・ポイント ……… 45

6. BIOSディスク定義テーブル ……… 49
- 6.1 ディスク・パラメータ・テーブル ……… 49
- 6.2 GENDEFによるテーブル・ジェネレーション ……… 53
- 6.3 GENDEFの出力 ……… 57

7. CP／M-86ブートストラップと改造手続き ……… 60
- 7.1 コールド・スタート・ロード・オペレーション ……… 60
- 7.2 CPM. SYSの構成 ……… 63

付　録

- A. セクターのブロッキングおよびデブロッキング ……… 65
- B. ランダム・アクセス・プログラムの例 ……… 72
- C. BOOT Rom ……… 79
- D. LDBIOS ……… 88
- E. BIOS ……… 95
- F. CBIOS ……… 110

第1章
CP／M-86システムの概要

1.1 CP／M-86の一般的特徴

CP／M-86は，最大1メガバイト（1,048,576）のメインメモリーをサポートし，かつCP／M-80のすべての能力を含んでいる．さらに，CP／M-86は，CP／M-80の以前のバージョンすべてとファイル互換性を保っている．CP／Mバージョン2のファイル構造は，16ドライブまで使え，そのドライブはそれぞれ8メガバイトまで使用することができ，CP／M-80とCP／M-86システムは，ファイル形式を変更することなしに，ファイルを交換できる．

CP／M-86は，CPM．SYSの中にあり，システム・イニシャライズのときにコールド・スタート・ローダーによってメモリにロードされる．コールド・スタート・ローダーは，システム・ディスクの最初の2つのトラックに書かれており，CPM．SYSは，次の3つのモジュールを含んでいる．

- コンソール・コマンド・プロセッサ（CCP）
- 基本ディスク・オペレーティング・システム（BDOS）
- ユーザが構成できる基本I／Oシステム（BIOS）

CCPとBDOSは，約10Kバイトを占有する．BIOSのサイズは，インプリメントにより大きさが異なる．オペレーティング・システムは，予約されたインタラプト・ロケーションより上のメモリで実行する．そのアドレス空間の残りは，BIOSテーブルで定義されたように8つの非連続領域に分割することができる．

CP／M-80とは異なり，CCPの領域は，トランジェント・プログラム・ロードにつづいたデータ領域としては使えない．すなわち，CP／M-86のすべてのモジュールは，常にメモリ上に存在し，ウォーム・スタートで再ロードされることはない．

CP／M-80と同様，CP／M-86はディスクからメモリ・イメージ・ファイルをロードし実行する．メモリ・イメージ・ファイルには，ヘッダ・レコードがはじめについている．ヘッダ・レコードについては，このドキュメントで述べてあり，正しくプログラムをロードし実行するのに必要な情報をもっている．CP／M-86でのメモリ・イメージ・ファイルは，ファイル・タイプCMDによって識別される．

CP／M-80と違ってCP／M-86は，システム・エントリまたはデフォルト変数に対する絶対ロケーションを使わない．BDOSエントリは，予約されたソフトウェア・インタラプトを通じて行われる．BIOSに対するエントリは，新しいBDOS呼び出しにより行われる．CP／M-80で低メモリで利用されていた2つの変数（デフォルトのディスク番号とI／Oバイト）は，それぞれCCPとBIOSに置かれる．

トランジェント・プログラムのデータ領域のデフォルトのFCBとコマンド・バッファのような絶対アドレスと関係のあるものは，最初のベース・ページの値を保持することによってCP／M-86では最小化できる．

ED，PIP，STAT，SUBMITのようなユーティリティ・プログラムは，CP／M-86とCP／M-80では，同じ方法でオペレートする．DDT-86は，CP／M-80で供給されるDDTとこのオペレーションにおいて類似しており，8086と8088のマシン・コードのインタラクティブ・デバッグをするものである．同様に，ASM-86は，Intel仕様のニーモニックを使って8086と8088のアセンブリ言語のプログラミングと開発を行うものである．

GENCMD（Generate CMD）ユーティリティは，CP／MのLOADプログラムと置きかわるものである．そして，それはASM-86またはIntelユーティリティによって使われたHexファイルを，CP／M-86の下で実行するためのメモリ・イメージ形式にあうように変換する．

さらに，LDCOPY（Loader Copy）プログラムは，SYSGENと置きかわり，システム・ディスクからコールド・スタート・ローダーをコピーし複製を作るのに使われる．さらに，LMCMDというGENCMDを変化させたものは，Intel LOC86ユーティリティからの出力をCMD形式に変換する．最後に，GENDEF（Generate DISKDEF）が，顧客のディスク・パラメータ・テーブル作成のために用意されている．

また，ASM-86，GECMD，LMCMD，GENDEFは，CP／M-80の下でのクロス開発のためにCOMファイル形式も供給している．

このマニュアルを通じて使われるいくつかの用語を下の表1-1に定義してある．

表1-1　CP／M-86用語

用語	意味
ニブル	4ビット　半バイト
バイト	8ビット値
ワード	16ビット値
ダブル・ワード	32ビット値
パラグラフ	連続した16バイト
パラグラフ境界	16で割切れるアドレス（最下位ニブルがゼロ）
セグメント	連続した最大64Kバイト
セグメント・レジスタ	CS，DS，ES，SSのいずれか
オフセット	セグメント・レジスタからの16ビットの移動量
グループ	セグメント・レジスタに相対的なリロケータブル・プログラム単位
アドレス	セグメント・レジスタ値とオフセット値から導びかれる有効メモリ・アドレス

グループは，メモリへロードする単位であるセグメント群からなっている．グループが64Kバイト以上のとき，はじめの64Kセグメントを越えたコードまたはデータがアクセスされるとき，セグメント・レジスタを管理するのは応用プログラムの責任である．

CP／M-86は，8つのプログラム・グループ（コード，データ，スタック，拡張および4つの補助グループ）をサポートする．コード，データ，スタックまたは拡張グループがロードされると，CP／M-86はそれぞれのセグメント・レジスタ（CS，DS，SS，ES）へそのグループのベースを設定する．また，CP／M-86は，4つの補助グループをもロードすることができる．トランジェント・プログラムは，CP／M-86によって格納された値を使う補助グループのロケーションをユーザのベース・ページにおいて管理する．

1.2　CP／M-80とCP／M-86との差異

CP／M-86の構造は，8080の応用プログラムを最小の努力で8086と8088プロセッサへ転換するために可能なかぎりCP／M-80に近づけ，同一のプログラム環境を提供している．この節では，CP／M-80とCP／M-86との間の明確な相違を指摘しているので，もし読者がすでにCP／M-80に精通しているならば，このマニュアル全体を読む必要はない．

この節で述べている言葉や概念は，このマニュアルのいたるところで詳細に説明されている．従って明確な定義と情報を述べている適切な節を目次から見つけることができる．

8086プロセッサの性質によって，CP／M-80とCP／M-86との間の基本的な差は，いろいろなリロケータブル・グループの管理にみいだすことができる．たとえCP／M-80がアブソリュート・メモリ・ロケーションを参照しなければならないとしても，CP／M-86の方は，8086プロセッサ固有の静的なリロケーションを利用する．

オペレーティング・システム自身は，インタラプト・ロケーションの上位（0400H）に通常にロードされ，リロケータブル・トランジェント・プログラムは，もっとも適合したメモリ領域にロードされる．しかし，オペレーティング・システムを変えることなしに任意のメモリ部分へCP／M-86をロードすることもできる（従って，CP／M-86にはMOVCPMユーティリティがない）．トランジェント・プログラムは，任意の予約されていない領域にロード，実行してもよい．

3つの一般的なメモリ・モデルが下に紹介されている．しかし，あなたが8080プログラムをCP／M-86へと変換するのであれば，8080モデルまたはスモール・モデルのいずれかを使うことができ，アドレスを増やしたくなったときあとでコンパクト・モデルに変更してやればよい．

GENCMD（3.2節に説明されている）を使って，Hexファイルから実行可能なプログラム・ファイルを作れる．GENCMDパラメータによって，あなたのプログラムが要求するメモリ・モデルを指定することができる．

CP／M-86自身は，8080モデルとして構成してある．これは，すべてのセグメント・レジスタがCP／M-86のベースにとられ，そして最も注目することとして顧客用に作られたBIOS（もちろん命令ニーモニックは変更したが）がCP／M-80のものと同一であることを意味している．事実，BIOSへエントリ・ポイントSETDMAB，GETSEGB，SETIOB，GETIOBが加えられただけである．あなたのウォーム・スタート・サブルーチンは，CP／M-86の下ではCCPとBDOSの再ロードを要求しないので単純なものとなる．もう1つ，IOBYTE機能をインプリメントするのであれば，自分のBIOSにその変数を定義しなければならない．

これらの変更点を考えてみると，8086のBIOSをインプリメントするのには,CP／M-80のBIOSを単に8086コードへ変換するだけでよい．

あなたがCP／Mバージョン2をインプリメントしたのであれば，すでにCP／M-86で正しく動作するディスク定義テーブルを持っている．あなたが別のディスク・ドライブをつけたいと思ってもよいし，またパフォーマンスをあげるためにセクタ・スキュー・ファクタで実験してもよい．そうするのであれば，CP／M-80のMACで使ったDISKDEFマクロと同じ機能を遂行する新しいGENDEFユーティリティを使えばよい．しかし，DISKDEFマクロよりもGENDEFの方がより多くの情報を与え，より良くエラー条件をチェックしている．

たとえ一般的にいってCP／M-80システムを生成するよりも，CP／M-86システムを生成することの方が易しいとしても，あなたが単密度のフロッピー・ディスクを使っているのであれば複雑なものとなる．

CP／M-86は，単密度のディスクの2トラックのシステム領域へ入れるにしてはあまりにも大きすぎる．従って，ブートストラップ・オペレーションは，CP／M-86をロードするのに，次の2つのステップを遂行しなければならない．最初のブートストラップは，コールド・スタート・ローダーをロードしなければならない．次にコールド・スタート・ローダーは，システム・ファイルからCP／M-86をロードする．コールド・スタート・ローダーには，INITエントリ・ポイントは別として，自分のCP／M-86　BIOSと同じLDBIOSが含まれている．

ローダーはディスクには書く必要がないから，あなたが望むのであればLDBIOSを単純化することができる．もし倍密度のディスクを持っているか，または単密度ディスク上に十分なトラックを予約することができるのならば，2段階のブートをせずにCP／M-86をロードできる．

ＢＤＯＳシステム・コールをするためには，予約されたソフトウェア・インタラプト＃224を使う．ＣＰ／Ｍ-80で見られたようなロケーション0005でのＢＤＯＳへのジャンプは，ＣＰ／Ｍ-86では用意されていない．しかし，オフセット0006でのアドレス・フィールドは用意されているから，このワード値を使った有効なメモリ量を決めるプログラムは変更せずにオペレートできる．

ＣＰ／Ｍ-80のＢＤＯＳの機能は,エントリ・パラメータとリターン・バリューに対して,8080のレジスタを使っている．ＣＰ／Ｍ-86のＢＤＯＳ機能は,8086のレジスタに対応しているテーブルを用いる．たとえば，8086レジスタＣＨとＣＬは，8080レジスタＢとＣに対応している．

表４-２におけるＢＤＯＳ機能コードのリストをみると，機能０，27，31とが 若干変更されている．また，いくつかの新しい機能が加えられているが，それらは今現存するプログラムには影響を与えない．

１つの主要な哲学上の差異としては，ＣＰ／Ｍ-80においてＢＤＯＳに与えられるすべてのアドレスが0000Ｈ～０ＦＦＦＦＨの範囲の単なる16ビット値になるということである．しかし,ＣＰ／Ｍ-86において，アドレスはあなたのデータ領域のベースに設定されるＤＳ(データ・セグメント）レジスタからちょうど16ビットのオフセットをもてる．あなたの手持ちのＣＰ／Ｍ-80プログラムをＣＰ／Ｍ-86環境下に変換するのであれば,あなたのデータ・セグメントは，64Ｋバイトより少ないはずである．この場合，ＤＳレジスタは初期ロードのあと変更する必要はない．従って，ＣＰ／Ｍ-80のすべてのアドレスは，単にＣＰ／Ｍ-86のＤＳに相対なオフセットとなる．

ＣＰ／Ｍ-80では，プログラムは，次の３つの方法で終了した．

・ ＣＣＰへ直接もどる
・ ＢＤＯＳファンクション０を呼ぶ
・ アブソリュート・ロケーション0000Ｈへ制御を渡す．

しかしＣＰ／Ｍ-86では，プログラムの終了にははじめの２つの方法だけをサポートしている．これは，0000Ｈへジャンプしたあと自動的にディスク・システムをリセットしないという副作用をもっている．その代り，ＣＣＰレベルでコントロールーＣを押すことにより達成される．

ＣＰ／Ｍ-86にはたくさんの新しいツールが用意されており,それらはプログラミングを単純なものとし，応用プログラムの作成能力をひろげることになるだろう．我々は，簡単にＣＰ／Ｍ-86の下でスタートできるようにそれを設計した．要するに，あなたがＣＰ／Ｍ-80からＣＰ／Ｍ-86へ変換するのであれば，8086機械コードへと変換する以外には主な変更はないであろう．さらに，ＣＰ／Ｍ-86のために設計するプログラムは，ネットワーク環境でディストリビューテッド・オペレーティング・システムを供給するＣＰ／ＮＥＴ-86と同様に，ＭＰ／Ｍ-86（マルチタスキング・オペレーティング・システム)のもとでも動させることができる．

第2章
CP／M-86のコマンド・セットアップと実行

この章では，CCP（Consol Command Processor）のオペレーション，トランジェント・プログラムの形式，CP／M-86メモリ・モデルとメモリ・イメージ形式を説明する．

2.1 CCPビルドイン・コマンドとトランジェント・コマンド

CP／M-86でのCCPのオペレーションは，CP／M-80とほぼ同じである．最初のコールド・スタートのとき，CP／Mのサインオン・メッセージがプリントされ，ドライブAが自動的にログインされる．それから，標準プロンプトがコンソールへ表 示される．

CP／M-86は，コンソールからの入力コマンド・ラインを待つ．そのコマンドには，ビルドイン・コマンド

DIR　DIRS　ERA　REN　TYPE　USER

のいずれかが含まれていてよい（SAVEは，DDT-86が同じ機能を実行するので，CP／M-86ではサポートされていないことに注意せよ）．

また，コマンド・ラインは〝コマンド・ファイル〟を表わすファイル・タイプ〝CMD〟をもつトランジェント・プログラムの名前で始まってもよい．CMDファイル・タイプは，CP／M-86の下で使われるトランジェント・コマンドとCP／M-80の下でオペレートするCOMファイルとを区別している．

CCPは，メモリ上に複数個のプログラムをおくことができ，バックグランド・タスクに対する道具を用意している．デバッガのようなトランジェント・プログラムは，自分自身の制御下で，実行すべき付加プログラムをロードしてもよい．従って，例えばバックグランド・プリンタ・スプーラは，はじめにロードされ，そのあとでDDT-86の実行をしてもよい．

DDT-86は，順次にデバッギング・セッションに対するテスト・プログラムをロードし，ブレーク・ポイントから次のブレーク・ポイントの間では，テスト・プログラムに制御を渡す．

CP／M-86は，プログラムがロードされた順序の記録を保持し，コントロール-Cが見つかるとCCPレベルで最も新らしい実行状態のプログラムを切り離す．DDT-86コマンド・レベルでのコントロール-Cは，DDT-86とそのテスト・プログラムとをアボートする．CCPレベルで2番目のコントロール-Cは，バックグランド・プリンタ・スプーラをアボートする．3番目のコントロール-Cは，ディスク・システムをリセットする．

コントロール-Cによるプログラム・アボートは，CP／M-80の場合と同じにディスク・システムをリセットしないことに注意せよ．メモリ上にプログラムが存在せず，コントロール-CがCCPコマンド・入力レベルで起こらないかぎり，ディスク・リセットは起こらない．

CP／M-86が，CCPまたは他のトランジェント・プログラムからトランジェント・プログラムをロードする要求を受けると，プログラムのメモリ要求をチェックする．十分なメモリがあると，CP／M-86は，プログラムに要求されたメモリ量を割当て，プログラムをロードする．ロードされた，そのプログラムは，BDOSからバッファ・スペースのための追加メモリを要求することができる．プログラムの実行が終了すると，CP／M-86は，プログラム・メモリ領域と追加されたバッファ・スペースとを解放する．

2.2 トランジェント・プログラムの実行モデル

トランジェント・プログラムで使われる3つの〝メモリ・モデル〟のいずれかが，セグメント・レジスタの初期値を決定し，CMDファイルのヘッダーの中に書かれる．3つのメモリ・モデルを下の表2-1にまとめた．

表2-1　CP／M-86メモリ・モデル

モ デ ル	グループの関係
8080モデル	コードとデータ・グループとが重複
スモール・モデル	コードとデータ・グループとが独立
コンパクト・モデル	3つ以上のグループが独立

8080モデルはコードとデータ領域が混在しているとき，CP／M-80から直接変換されるプログラムをサポートする．8080モデルは，コード，データ，スタック領域はすべて1つのグループからなっている．

セグメント・レジスタは，このグループを含む領域の開始アドレスを初期値として与えられる．しかし，セグメント・レジスタは実行中に，応用プログラムによって管理されるから，そのコード・グループ内の多数のセグメントをアドレス付けできる．

スモール・モデルは，Intelによって定義されたものと同じであり，そのプログラムは独立したコード・グループとデータ・グループとからなっている．スモール・モデルは，コードとデータとが簡単に分離できるようなCP／M-80のプログラムに使用するのが適当である．コード・グループとデータ・グループは，しばしば1つの64Kバイト・セグメントからなる（しかし制限はされないが）ことに注意せよ．

コンパクト・モデルは，拡張，スタックまたは補助グループのどれかがプログラムの中に存在するときに使われる．各グループは，1つ以上のセグメントからなっていてもよい．しかし，いずれかのグループのサイズが1セグメントをこえるか，または補助グループがあるのであれば，応用プログラムではすべてのコードとデータ領域とをアドレス付けするために対応するセグメント・レジスタを実行中に管理しなければならない．

3つのモデルには主として，トランジェント・プログラムがローディングされたときのセグメント・レジスタの初期値設定方法に違いがある．オペレーティング・システム・プログラムのロード機能は，次の章に述べられているように，プログラム・グループの扱い方を調べることによって，トランジェント・プログラムで使われるメモリ・モデルを決定する．

2.3　8080メモリ・モデル

8080モデルは，トランジェント・プログラムがコード・グループだけを含んでいるものと仮定している．この場合に，ＣＳ，ＤＳ，ＥＳレジスタは，コード・グループの先頭に初期化される．ＳＳ，ＳＰレジスタは，ＣＣＰの中の96バイト・スタック領域に設定される．インストラクション・ポインタ・レジスタ（ＩＰ）は，ＣＰ／Ｍ-80と同様に，100Ｈに設定される．従って，ベース・ページの値はコード・グループの始めとなる．プログラム・ロード後，8080モデルは，図２-１に示したようになる．ここで低アドレスは，図の上方である．

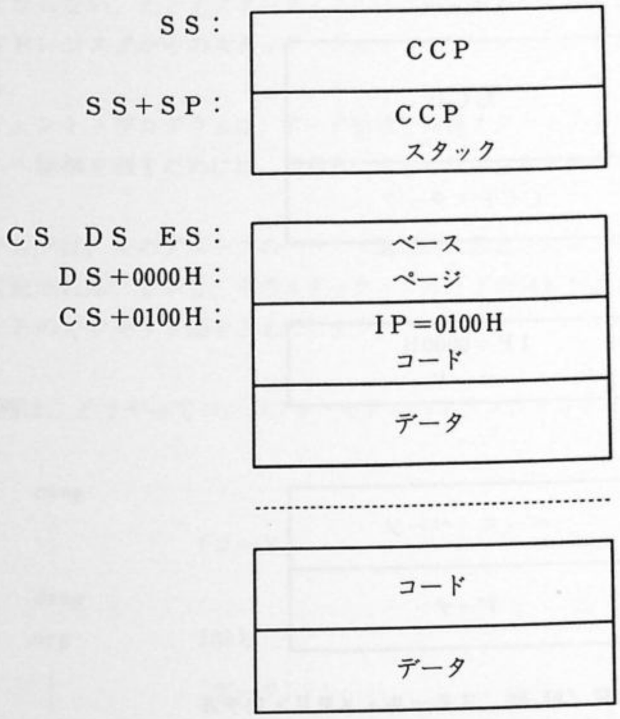


図２-１　ＣＰ／Ｍ-86　8080メモリ・モデル

コードとデータ領域が混在していると，見わけがつかない．下に述べる〝ベース・ページ〟の値は，ＣＰ／Ｍ-80と同一であり，8080，8085，Ｚ80コードから8086と8088環境へと単純に変換できる．次のＡＳＭ-86の例は，どのようにして8080モデルのトランジェント・プログラムをコードするのかを示している．

```
        cseg
        org     100h
          ⋮     (コード)
endcs   equ       $
        dseg
        org     endcsからのオフセット
          .
          .     (データ)
          .
        end
```

2.4 スモール・メモリ・モデル

スモール・モデルは，トランジェント・プログラムがコードとデータ・グループの両方を含むと仮定している．（ASMでは，コードはすべてCSEGディレクティブのあとに生成される．データは，DSEGディレクティブのあとに定義され，コード・セグメントと独立なデータ・セグメントのオリジンをもつ）．

このモデルでは，CSは，コード・グループの先頭に設定される．DSとESは，データ・グループの先頭に設定される．SSとSPレジスタは，図2-2にみられるように，CCPのスタック領域にある．

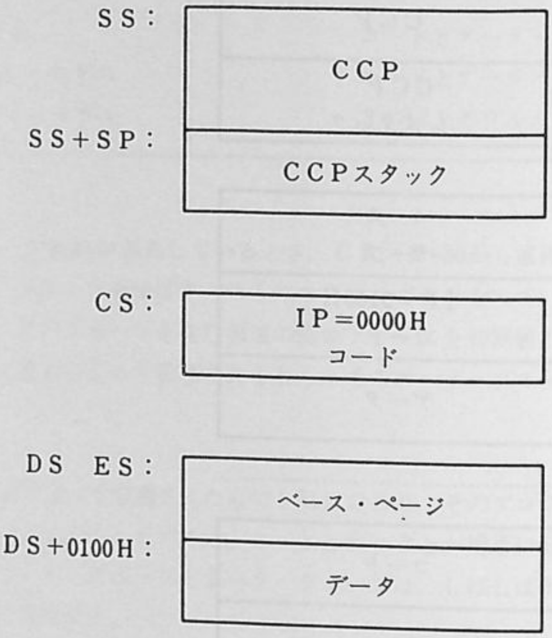


図2-2　CP／M-86　スモール・メモリ・モデル

マシン・コードは，CS＋0000Hではじまる．"ベース・ページ"の値は，DS＋0000Hではじまる．データ領域は，DS＋0100Hではじまる．

次のASM-86の例は，スモール・モデルのトランジェント・プログラムをどのようにコードするかを示している．

```
cseg
  ⋮        （コード）
dseg
org        100h
  ⋮        （データ）
end
```

2.5 コンパクト・メモリ・モデル

コンパクト・モデルは，コードとデータ・グループが残りのスタック，拡張または補助グループのいずれかと一緒に存在すると仮定する．この場合，ＣＳ，ＤＳ，ＥＳレジスタは，それぞれの領域のベース・アドレスに設定される．

図２－３は，コンパクト・モデルのセグメント・レジスタの初期構成を示している．ＣＣＰによって，ベース・ページに置かれた初期値からローディングすることによりセグメント・レジスタの値は，実行中にプログラムで変更することができる．従って，全体のメモリ・スペースにアクセスできる．

トランジェント・プログラムがスタック・グループをスタック領域として使おうとすれば，ＳＳとＳＰレジスタはエントリに設定されなければならない．たとえスタック・グループが定義されようとも，ＳＳとＳＰレジスタは，ＣＣＰ領域の中に残っている．ＳＳとＳＰレジスタがそのスタック・グループをアドレス付けするために設定されなければならないとしても，２つの矛盾が存在する．

第１は，トランジェント・プログラムは，データ領域と同様スタック・グループを使っていてもよい．この場合，トランジェント・プログラムへ制御を渡すためには，ＣＣＰによって使われる Far Call 命令は，スタック領域の中にデータを上書きすることができる．

第２に，ＳＳレジスタは，そのグループのベースへ論理的に設定される．そして，ＳＰレジスタは，グループの終りまでのオフセットとして設定される．しかし，そのスタック・グループが64Ｋをこえると，ベースからそのグループの終りまでのアドレス範囲は16ビットのオフセット値をこえてしまう．

次のＡＳＭ-86の例は，どうやってコンパクト・モデルのトランジェント・プログラムをコードするかを示す．

```
cseg
  :
  :           （コード）
dseg
org           100 h
  :
  :           （データ）
eseg
  :
  :           （他のデータ）
sseg
  :
  :           （スタック領域）
end
```

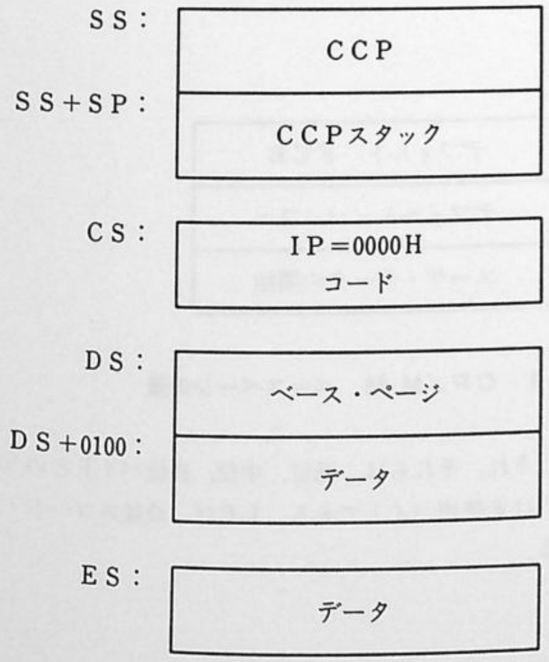


図２-３　CP／M-86　コンパクト・メモリ・モデル

2.6 ベース・ページの初期化

CP／M-80と同様，CP／M-86ベース・ページにはCCPによって初期化され，トランジェント・プログラムで使用されるデフォルト値とロケーションが入っている．ベース・ページは，DSレジスタに対するオフセット0000Hから00FFHまでの領域を占有する．CP／M-86に対するベース・ページ内の値は，CP／M-80のそれを含んでおり，図2－4で示されるように同じ相対立置にある．

DS+0000	LC0	LC1	LC2
0003	BC0	BC1	M80
0006	LD0	LD1	LD2
0009	BD0	BD1	XXX
000C	LE0	LE1	LE2
000F	BE0	BE1	XXX
0012	LS0	LS1	LS2
0015	BS0	BS1	XXX
0018	LX0	LX1	LX2
001B	BX0	BX1	XXX
001E	LX0	LX1	LX2
0021	BX0	BX1	XXX
0024	LX0	LX1	LX2
0027	BX0	BX1	XXX
002A	LX0	LX1	LX2
002D	BX0	BX1	XXX
0030	現在未使用		
………			
005B			
005C	デフォルト　FCB		
0080	デフォルト・バッファ		
0100	ユーザ・データの開始		

図2－4　CP／M-86　ベースページの値

各バイトは，0，1，2でインデックス付けされ，それらは，低位，中位，高位バイトという標準的なIntelのストレージ協約に対応している．図2－4の中の"XXX"は未使用バイトである．LCは，最後のコード・グループのロケーション（24ビット，高位4ビットはゼロに等しい）である．

8080モデルでは，ＬＣの低位バイト（ＬＣ０とＬＣ１）は，決して０ＦＦＦＦＨを越えない．そして，高位バイト(ＬＣ２)はつねにゼロである．ＢＣは，コード・グループのパラグラフ・アドレス（16ビット）である．ＬＤとＢＤは，データ・グループの最後の位置とパラグラフ・ベースとを規定している．最後の位置は，グループの長さより１バイト少ない．バイトＬＤ０とＬＤ１がＣＰ／M-80とＣＰ／M-86との両方ともベース・ページの同じ相対位置にあらわれることは注目に価する．従って，プログラムの変換作業も簡単に行える．M80のバイトは，8080メモリ・モデルが使われていると１に等しくなっている．ＬＥとＢＥは，オプションの拡張グループの長さとパラグラフ・ベースとを規定している．ＬＳとＢＳは，オプションのスタック・グループの長さとベースを与える．ＬＸとＢＸバイトは，コンパクト・メモリ・モデルを使って実行するプログラムに対して要求される４つのオプションの独立なグループの集りに対応している．これらの記述子に対する初期値は，次の章で説明されているメモリ・イメージ・ファイルのヘッダ・レコードから取出される．

2.7　トランジェント・プログラムのロードと出口

ＣＰ／M-80と同様，ＣＣＰはコマンドから２つのファイル名を分解し，ＤＳレジスタに相対的なベース・ページの中のロケーション005ＣＨと006ＣＨにある独特に形式化されたＦＣＢに置く．

ＣＰ／M-80の下では，デフォルトのＤＭＡアドレスは，ベース・ページの0080Ｈに初期化されている．8086と8088プロセッサは，メモリがセグメント化されているために，ＤＭＡアドレスは，２つの部分（ＤＭＡセグメント・アドレスとＤＭＡオフセット）に分けられる．故に，ＣＰ／M-86の下では，デフォルトのＤＭＡベースは，ＤＳの値に初期化され，ＤＭＡオフセットは0080Ｈに初期化される．従って，ＣＰ／M-80とＣＰ／M-86は同じ方法でオペレートする．両方ともデフォルトＤＭＡバッファがベース・ページの後の半分を占有するものとみなされる．

ＣＣＰは，8086 〝Far Call〟 を通じてトランジェント・プログラムへ制御を渡す．トランジェント・プログラムは，96バイトのＣＣＰスタックを使ってもよいので，この場合には 〝Far Rturn〟 を実行することによってプログラムの終了の時ＣＣＰへ直接戻ることができる．また，ＢＤＯＳ機能０の実行によりプログラムを終了することもできる．機能０は，メモリからプログラムを移動したり，またはメモリ・アロケーションの状態を変化させることなくプログラムを終了すること ができる（図４-２参照)．

オペレータは，ライン編集入力のときプログラムがＢＤＯＳ機能０を実行するのと同じ効果をもつ，コントロール-Ｃをタイプすることによってプログラムの実行を終了してもよい.ＣＰ／M-80のオペレーションと違って,プログラム終了時にディスクはリセットされず，ＣＣＰとＢＤＯＳモジュールはディスクから再ロードされない．

第3章
コマンド（CMD）ファイルの生成

前述したように，CP／M-86ではGENCMDとLMCMDという2つのユーティリティ・プログラムが提供されている．それはCP／M-86の下で実行できるようなCMDメモリ・イメージ・ファイルを作るのに使われる．

GENCMDは入力としてIntel 8086の〝hex″形式ファイルを受けつける．LMCMDは，Intel　Lモジュール・ファイルを読む．それは，Intel　LOC　86オブジェクト・コード・ロケータ・ユーティリティから出力される．GENCMDは，Digital Research　ASM-86アセンブラとIntel OH86ユーティリティからの出力を処理するのに使う．一方，LMCMDは，Intelと互換性をもつように開発されたソフトウェアがCP／M-86オペレーションを目標としたプログラムの生成に対して利用できるときに使われる．

3.1　Intel 8086　Hexファイル形式

GENCMDの入力は，Digital Research　ASM-86アセンブラと，標準Intel　OH86ユーティリティ・プログラム（Intelドキュメント＃9800639-03〝MCS-86　Software Development Utilities Operating Instructions for ISIS-II　Users″を参照せよ）の両方によって作られたIntel〝hex″形式である．

GENCMDで作られたCMDファイルは，CMDファイルをロードし実行するのに必要なメモリ・モデルとメモリ・サイズを定義したヘッダ・レコードを含んでいる．

Intel〝hex″ファイルは，次の形式をしたASCIIレコードの並びからなっている．

:	l	l	a	a	a	a	t	t	d	d	d	・・・	d	c	c

レコードの始めは，ASCIIのコロンで示され，後続のそれぞれの文字位置は0～9，A～Fの範囲のASCII16進数を含んでいる．

表3-1にそのフィールドを定義している．

表3-1　Intel hexフィールドの定義

フィールド	内　容
ll	レコード長　00～FF（10進で，0～255）
aaaa	ロード・アドレス
tt	レコードのタイプ
	00　データ・レコード 現在のベース．パラグラフからオフセットaaaaで始まるようにロードする．
	01　ファイルの終り，CC＝FF
	02　拡張アドレス． aaaaは後続のデータ・レコードに対するパラグラフ・ベースである．
	03　スタート・アドレスはaaaaである．（省略すると，IPは使用中のメモリ・モデルによって設定される）
	次のものは，ASM-86だけから出力される．
	81　00と同じ，データはコード・セグメントに属している．
	82　00と同じ，データはデータ・セグメントに属している．

	83　00と同じ，データはスタック・セグメントに属している． 84　00と同じ，データは拡張セグメントに属している． 85　アブソリュート・コード・セグメントに対するパラグラフ・アドレスである． 86　アブソリュート・データ・セグメントに対するパラグラフ・アドレスである． 87　アブソリュート・スタック・セグメントに対するパラグラフ・アドレスである． 88　アブソリュート拡張セグメントに対するパラグラフ・アドレスである．
d	データ・バイト
cc	チェック・サム (00～前の文字までの和)

各レコードのコロンの前の文字は，すべて無視される．(それ以上の hex ファイル形式の情報は，ＡＳＭ-86ユーザース・ガイドと〝ＭＳＣ-86 Absolute Object file Format〟と題されたIntelのドキュメント＃9800821Ａとに説明されている)．

3.2 GENCMDのオペレーション

ＧＥＮＣＭＤユーティリティは，ＣＣＰレベルで，次のようにタイプして呼び出す．

ＧＥＮＣＭＤ　filename parameter-list

ここで，filenameは，H86ファイル・タイプをもつ（明記されてはいない）16進入力ファイルに対応する．

ＧＥＮＣＭＤは，特に8080メモリ・モデルを識別し，そして各セグメント・グループのメモリ要求を記述するための付加パラメータを受けつける．

ＧＥＮＣＭＤのパラメータは，filenameのあとへ続けて並べられる．そして，上のコマンド・ラインに示したように，パラメータ・リストはキーワードと値とをコンマまたはブランクで区切って並べたものである．キーワードには，次のものがある．

8080　CODE　DATA　EXTRA　STACK　X1　X2　X3　X4

キーワードの8080は，コード・グループだけとされ，ＢＤＯＳロード機能は実行のために8080メモリ・モデルを設定する．従って，1つのセグメントの中にコードとデータとが混在している．このコマンド形式は，次の形をしている．

ＧＥＮＣＭＤ　filename　8080

残りのキーワードは，filenameまたは8080オプションの後にくる．そして前節で定義したセグメント・グループと１対１に対応した各セグメント・グループの明確なメモリ要求を定める．それぞれの場合に，各グループに対応する値は，カギ・カッコで囲こまれ，コンマで区切られる．各値はパラグラフ・アドレスを表わす16進数であるか，または１つの文字を付したhhhhで示されるパラグラフ単位のセグメントの長さである．各値の意味は次のようになっている．

Ahhhh	アブソリュート・ロケーション hhhh にグループをロードする．
Bhhhh	グループは，hex ファイルの hhhh で始まる．
Mhhhh	グループは，最小 hhhh×16バイト必要とする．
Xhhhh	グループは，最大 hhhh×16 バイトをアドレス付けすることができる．

一般に，ＣＭＤファイルのヘッダの値は，hex ファイルから直接取り出され，上に示されたパラメータを含める必要はない．しかし，次のような情況のときには，ＧＥＮＣＭＤパラメータを必要とする．

8080キーワードは，ソース・プログラムでＣＳＥＧとＤＳＥＧディレクティブを使うことなしに，コードとデータとが１つの64Ｋセグメント内に混合されているとき，ＡＳＭ-86が8080プログラムを8086／8088環境へ変換しようとするときは常に必要である．

アブソリュート・アドレス（A値）は，アブソリュート・ロケーションに位置づけられるグループに対して与えられなければならない．

一般に，この値はCP／M-86が要求するメモリ領域が利用可能であることを保証することができないため指定できない．この場合，CMDファイルはロードされない．

B値は，GENCMDがIntelのOH86によって作られたhexファイルまたは1つ以上のグループを含む同じようなユーティリティ・プログラムを処理するときに使われる．

コード，データ，拡張，スタックまたは補助グループを識別するための情報をもたないデータ・レコードの並びから，OH86の出力は構成される．この場合，B値はキーワードによって指定されたグループの開始アドレスを示している．GENCMDは，その指定されたグループへ，このアドレスに従ってデータをロードさせる．従って，B値はセグメント情報がhexファイルに含まれていないとき，コードとデータ・セグメントとの間の境界を明示するために使われる．

ASM-86によって作られるファイルは,セグメント情報がhexファイルに含まれているから，B値は使用しなくてよい．

最小メモリ値（M値）は，hexレコードが指定されたグループに対して最小メモリ要求を定義しないときだけ使われる．一般に，コード・グループのサイズは，その領域にロードされるデータ・レコードによって正確に決定される．すなわち，そのグループに要求される全体のスペースは，最低と最高のデータ・バイト・アドレスとの間の範囲によって定義される．しかし，データ・グループはそのグループの終りに初期化されていない領域を含んでもよい．だから，データ・レコードは最高位で参照されるデータ項目を定義するファイルには存在しない．

そのデータ・グループの最高位アドレスは，最後のデータ項目として，〝DB　0〟を入れることによって，ソース・プログラム内に定義することができる．

また，M値は，そのグループの終りに追加スペースを取るために指定することができる．同様に，スタック，拡張，補助グループのサイズは，そのグループ内の最高位アドレスがhexファイル内のデータ・レコードによって暗黙的に定義されるほかは，M値を使って定義しなければならない．

X値で与えられる最大メモリ・サイズは，一般には追加される自由メモリがI／Oバッファやシンボル・テーブルとして使われるときに必要となる．

データ領域のサイズが固定であれば，Xパラメータを指定してはならない．このとき，X値はM値と同じとみなされる．値XFFFFは，利用可能な最大メモリ領域を取る．しかし，もし利用するのであれば，トランジェント・プログラムは，3バイト長のフィールドがこのグループのためにベース・ページ内に作られると知っていなければならない．ここで，そのフィールドは，最高位バイトがゼロでなくてもよい．

CP／M-80から直接変換されるプログラムまたはアドレス・バッファへの2バイトのポインタを使うプログラムが，0FFF0Hバイトの最大長のアロケーションをつくりだすときにはこの値をXFFFまたはそれ以下に制限すべきである．

次のGENCMDコマンド・ラインは，ファイルX．H86を独特のヘッダ・レコードをもつファイルX．CMDへ変換する．

```
gencmd  X  code [a40]  data [m30, Xfff]
```

この場合そのコード・グループは，パラグラフ・アドレスを40H，言いかえればバイト・アドレスは400Hとなり，そのデータ・グループは，最大300Hバイトを要求するが，利用可能であれば0FFF0Hバイトまで使うことができる．

ファイルY．H86は，散在しているセグメント情報をもたないIntel hexレコードを含み，ドライブBにあるものとする．そのコマンド

```
gencmd  b：y  data [b30, m20] extra [b50] stack [m40]  X1 [m40]
```

は，コード・セグメントに対してはアドレス0000Hで始まり，データ・セグメントに対しては300Hとしてアロケートされたレコードを選らぶことによって，ドライブB上にY．CMDを作りだす．拡張セグメントは,500Hで始まるレコードから詰めこまれる．

スタックと補助セグメント＃1は，最小の400Hバイトをそれぞれ要求する非初期化領域である．この例では，データ領域は最小の200Hバイト必要としている．

もしDigital Research　ASM-86アセンブラが使われると，B値を含める必要がないことに再び注意して下さい．

3.3 LMCMDのオペレーション

LMCMDユーティリティは，GENCMDとほぼ同じ方法でオペレートする．ただしLMCMDの入力が Intel L-モジュール・ファイルを受けつけることだけが異っている．

L-モジュール形式の主な利点は，そのファイルがグループのデータ・セグメントの開始アドレスと言うような，GENCMDではパラメータとして別の方法で与える値を定義している内部的レコード情報を含んでいる点である．しかし，現在この形式を使っているのは，標準 Intel 開発パッケージの中で言語プロセッサだけである．おそらく，いろいろな独立ベンダーが将来この形式を利用するようになるだろう．

3.4 コマンド（CMD）ファイルの形式

GENCMDとLMCMDとによって作られたCMDファイルは，128バイトのヘッダ・レコードと，その後につづいたメモリ・イメージから構成される．通常の環境の下では，そのヘッダ・レコードの形式はプログラマにとっては重要なことではない．しかし，完全性をきたす意味で，このレコードのフィールドを図3-1に示してある．

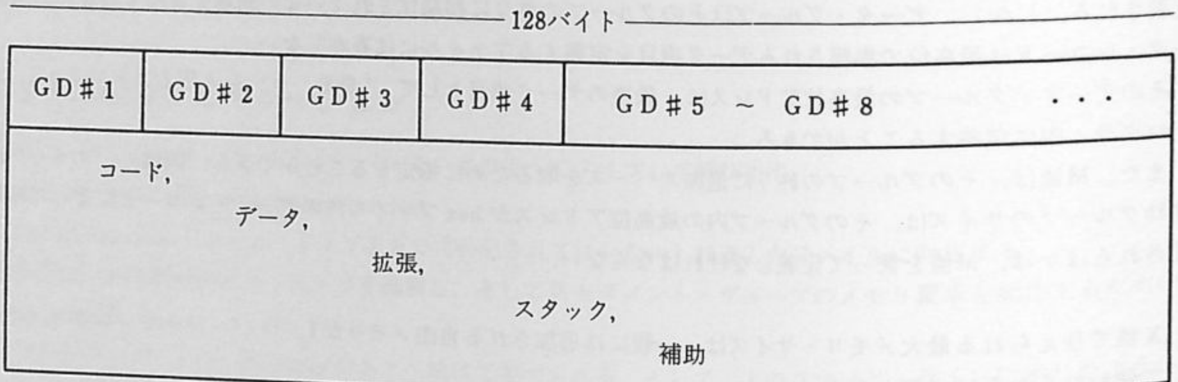


図3-1　CMDファイルのヘッダ形式

図3-1で，GD＃2～GD＃8は，〝グループ記述子〟を表わしている．各グループ記述子は，それぞれロードされるプログラム単位に対応しており，次のフィールドをもつ．

8ビット	16ビット	16ビット	16ビット	16ビット
G-様式	G-長さ	A-ベース	G-最小	G-最大

G様式は，グループの形式を示し，次に記述子がないときはゼロの値である．

G様式がゼロでないとき，8ビットの値は2つのフィールドに分けられる．

G様式

4ビット	4ビット
XXXX	G-タイプ

Gタイプは，グループ記述子のタイプを決定する．正しいグループ記述子は，表3-2で示すように，1～9の範囲のGタイプをもつ．

表3-2　グループ記述子

Gタイプ	グループ・タイプ
1	コード・グループ
2	データ・グループ
3	拡張グループ
4	スタック・グループ
5	補助グループ　＃1
6	補助グループ　＃2
7	補助グループ　＃3
8	補助グループ　＃4
9	分割されたコード・グループ
10〜14	未使用，予約済み
15	追加タイプに対するエスケープ・コード

グループ記述子の残りの値はすべて，16バイトのパラグラフ単位（20ビット・アドレスをつくるのに低位ニブルを0と仮定する）の増分で与えられる．

G-長さは，グループ内のパラグラフの個数である．例えば，0080HのG-長さが与えられていると，グループのサイズは，00800H＝2048Dバイトである．

A-ベースは，非再配置的グループに対するベースのパラグラフ・アドレスを定義している．

G-最小とG-最大は，グループに配置するメモリ領域の最小と最大サイズとを定義する．

G-タイプ9は，MP／M-86とCP／M-86の将来バージョンの下で使うための〝pure〞コード・グループを示す．

ヘッダ・レコードで記述されたメモリ・モデルは，グループ記述子によって暗黙的に決定される．8080メモリ・モデルは，独立のデータ・グループが指定されていないから，コード・グループがあるとだけ仮定される．

スモール・モデルは，コードとデータ・グループの両方があり，追加グループ記述子がないときのみ適用される．

その他，コンパクト・モデルは，CMDファイルがロードされるときに推定される．

第4章
基本ディスク・オペレーティング・システムの機能

ここでは，トランジェント・プログラムでCP/M-86のBDOSとBIOSの機能を利用するためのインターフェイスについて説明する．

BDOSの呼び出しはCP/M-80バージョン2とほとんど一致しているので，CP/M-80用のプログラムをCP/M-86に移行する作業は容易である．まずBDOSへ入る時と戻る時の条件について説明し，次に個々のBDOS機能の呼び出し方について述べる．

4.1 BDOSのパラメータと機能コード

BDOSへ制御を移すのは，8086ソフトウェア・インタラプト#224によって行なわれ，これはCP/M-86やMP/M-86による利用するために予じめインテル社で用意されている．機能コードCLレジスタに，バイトパラメータはDLに，ワード・パラメータはDXレジスタに格納して受け渡しが行なわれる．

BDOS機能から戻るときには，1バイトの値はALレジスタに，1ワードの値はAXレジスタとBXレジスタの両方に，2ワード長の値はESレジスタとBXレジスタに入れて戻る．すべてのセグメント・レジスタは，ESレジスタを除いてBDOSへ入る時点で値が保存され，戻る時に再格納される．(これはPL/M-86と同様である)．

受け渡されるパラメータの入出力の一覧を表4-1に示す．

表4-1 BDOSパラメータの一覧表

BDOSエントリ・レジスタ	BDOSリターン・レジスタ
CL 機能コード	バイトはALレジスタ
DL バイト・パラメータ	ワードはAXレジスタとBXレジスタのいずれか
DX ワード・バラメータ	
DS データ・セグメント	2ワード長はBXレジスタにオフセットとESレジスタにセグメント

注 意

CP/M-80のBDOSは，種々の機能の入力として"情報番地"を要求している．一般的にこの番地は，バッファまたはシステム・コールによって使用されるファイル・コントロール・ブロックである．一方CP/M-86では，その情報番地は現在のDSレジスタの内容とDXレジスタによるオフセットとの組合せにより与えられる．これはDSレジスタに適当な値が格納されているという仮定のもとに，CP/M-86システムでのDXレジスタはCP/M-80システムの2バイトのDEレジスタと同じ働きをしている．

1個のデータ・セグメントだけを使用しているプログラム（CP/M-80からCP/M-86へ変換しようとしているプログラムの場合に当る）このような特定の問題の場合ではなく，データ・グループが1セグメントを越える時に，DSレジスタにプログラムで参照しているデータ領域を含むセグメントの値を格納しなければならない．また，範囲外の機能呼び出しはゼロの値が返される．

CP/M-86で利用できる機能コードを表4.2に示す．(*)の付いている機能コードは，CP/M-80オペレーティング・システムのバージョン2.0に対して変更または追加のあったものである．

表4-2　CP／M-86オペレーティング・システムBDOSの機能

F#	機　能	F#	機　能
0＊	システム・リセット	25	選択されているディスク番号の取り出し
1	コンソールからの入力	26	DMAアドレスのセット
2	コンソールへの出力	27＊	アロケーション・アドレスの取り出し
3	リーダからの入力	28	書き込み保護ディスク
4	パンチへの出力	29	R／Oベクトルの取り出し
5	リストへの出力	30	ファイル属性のセット
6＊	コンソールによる直接入出力	31＊	ディスク・パラメータ・アドレスの取り出し
7	I／Oバイトの取り出し	32	ユーザ・コードのセットまたは取り出し
8	I／Oバイトのセット	33	ランダムな読み出し
9	文字列のプリント	34	ランダムな書き込み
10	コンソール・バッファへの読み込み	35	ファイルの大きさの計算
11	コンソール・ステータスの取り出し	36	ランダム・レコード番のセット
12	バージョン番号の取り出し	37＊	ドライブのセット
13	ディスク・システムのリセット	40	ランダムな書き込み（ゼロ・フィル）
14	ディスクの選択	50＊	ダイレクトBIOSコール
15	ファイルのオープン	51＊	DMAセグメント・ベースのセット
15	ファイルのクローズ	52＊	DMAセグメント・ベースの獲得
17	最初のデータのサーチ	53＊	利用可能最大メモリの獲得
18	次のデータのサーチ	54＊	絶対ロケーションにおける最大メモリ
19	ファイルの削除	55＊	メモリ領域の獲得
20	シーケンシャルな読み出し	56＊	絶対メモリ領域の獲得
21	シーケンシャルな書き込み	57	メモリ領域の解放
22	ファイルの作成	58	全メモリの解放
23	ファイル名のつけ直し	59	プログラムのロード
24	ログイン・ベクトルの取り出し		

個々のBDOSの機能について，以下3つのセクションに分けて説明してある．それらは簡単なBDOS機能や，ファイル処理，そしてメモリ管理やプログラム・ローディングに関する拡張処理である．

4.2　単純なＢＤＯＳ機能の呼び出し

まずＢＤＯＳの機能コード０から12まではシステム・リセットや１文字の入出力など簡単な機能を遂行する．

入力パラメータ　｜機能コード　0　システム・リセット｜　終了時の状態

ＣＬ：00Ｈ
ＤＬ：アボート・コード

システム・リセット機能は，ＣＣＰコマンド・レベルでＣＰ／Ｍオペレーティング・システムへ制御を戻す．レジスタＤＬに格納されているアボード・コードは２つの値をもつ．もし，ＤＬ＝00Ｈなら，現在動いているプログラムが終了し，ＣＣＰへ制御を戻す．その際，プログラムが存在しているメモリ・エリアは解放される．もし，ＤＬが01Ｈなら，そのプログラムはメモリ内に残りメモリ配置状態はそのまま変らずに残っている．

入力パラメータ　｜機能コード：1　コンソールからの入力｜　終了時の状態

ＣＬ：01Ｈ　　ＡＬ：ＡＳＣＩＩ文字

コンソールからの入力機能では，論理コンソール（ＣＯＮＳＯＬＥ）からの入力をレジスタＡＬに格納する．グラフィック文字，キャリッジ・リターン（復帰），ライン・フィード（改行）およびバック・スペース（コントロール-Ｈ）入力についてはコンソール画面上でのみ描かれる．タブ（コントロール-Ｉ）については８文字だけ，桁を進める．

コンソールからのタイプ入力がないと，ＢＤＯＳによっては呼び出しを行なったプログラムへ戻らないので，プログラムの実行を一時中止するのに用いられる．

入力パラメータ　｜機能コード　2　コンソールへの出力｜　終了時の状態

ＣＬ：02Ｈ
ＤＬ：ＡＳＣＩＩ文字

コンソールへの出力機能は，レジスタＤＬに格納されているＡＳＣＩＩ文字が論理コンソールへ出力される．タブは８文字文だけ桁を進め，スクロール（コントロール-Ｓ）の開始・停止の点検を行なうことができる．

入力パラメータ　｜機能コード　3　リーダからの入力｜　終了時の状態

ＣＬ：03Ｈ　　ＡＬ：ＡＳＣＩＩ文字

リーダからの入力機能は，論理リーダ（ＲＥＡＤＥＲ）からの入力をレジスタＡＬに格納する．リーダからの入力がないと，呼び出しを行なったプログラムへは戻らない．

入力パラメータ　｜機能コード　4　パンチへの出力｜　終了時の状態

ＣＬ：04Ｈ
ＤＬ：ＡＳＣＩＩ文字

パンチへの出力機能は，レジスタＤＬに格納されている文字を論理パンチ装置（ＰＵＮＣＨ）へ出力する．

入力パラメータ	機能コード　5 リストへの出力	終了時の状態
CL：05H DL：ASCII文字		

リストへの出力機能は，レジスタDLに格納されている文字を論理リスト装置（LIST）へ出力する．

入力パラメータ	機能コード　6 コンソールによる直接入出力	終了時の状態
CL：06H DL：0FFH（入力／ステータス） 　　0FEH（ステータス） 　　文字　　（出力のとき）		AL：文字またはステータス（入力のとき） 　　なし（出力のとき）

コンソールによる直接入出力機能は，CP／M-86オペレーティング・システムのもとで，コンソールとの間で直接入出力を行なう特別な応用プログラムを書くための機能である．この機能を用いる時には，CP／M-86のもとで行なわれていたコントロール-S，およびコントロール-Pのようなコントロール機能は行なわれない．また，これまでのCP／M-80オペレーティング・システムのリリースでは，このコンソールによる直接入出力機能を基本I／Oシステム（BIOS）と共に働かせるには，コンソールによる直接入出力機能を基本ディスク・オペレーティングシステム（BDOS）のもとで働かせるように変更しなければならない．将来のMP／M，またはCP／Mのもとでは，何も変更しなくてもコンソールによる直接入出力機能を行なうことができる．

機能コード6に於いては，レジスタDLにそれぞれ次の様な値を格納する．

(1) コンソールからの入力／ステータス を要求するときには，16進でFFを格納する．
(2) コンソールの状能をしらべるときには，16進でFEを格納する．
(3) それ以外のときは，コンソール（論理コンソール装置）へASCII文字を出力する．

レジスタDLにFFを格納すると，この機能によりまずBIOSのコンソールからの入力を開始し，呼び出したプログラムに戻った時にはレジスタALにコンソールから入力した文字が格納されている．また，レジスタDLにFEを格納すると，戻る時に入力文字がない時には，レジスタAL＝00となり，そうでなければAL＝FFとなる．

レジスタDLにFEやFF以外のものを格納した時には，レジスタDLに格納されているASCII文字をコンソールへ出力する．

入力パラメータ	機能コード　7 I／Oバイトの取り出し	終了時の状態
CL：07H		AL：入出力データ・バイトの値

I／Oバイト（IOBYTE）の取り出し機能は，IOBYTEに現在格納されている値をレジスタALに格納する．IOBYTEは，論理的な装置としてのCONSOLE，READER，PUNCHやLISTの割り当てを定義している．IOBYTE機能はBIOSの中に用意してある．

入力パラメータ	機能コード　8 I／Oバイトのセット	終了時の状態

ＣＬ：08H
ＤＬ：I／Oバイトの値

I／Oバイト（IOBYTE）のセット機能は，レジスタDLに格納されている値をIOBYTEにセットする．この機能により，トラジェント・プログラムは論理装置であるCONSOLE，READER，PUNCH，LISTの割り当ての定義にIOBYTEを使用できる．

入出力パラメータ	機能コード　9 文字列のプリント	終了時の状態

ＣＬ：09H
ＤＸ：文字列の格納されているオフセット

文字列のプリント機能は，レジスタDXによって示される文字列の格納されている番地に記憶されている文字列を論理コンソール装置（CONSOLE）へ表示する．コンソールへの表示は文字列の最後を示すドル記号（＄）まで出力する．機能コード2と同じように，タブは8文字分だけ桁を進め，またスクロールの開始・停止およびプリント出力の開始・停止の点検を行なうことができる．

入出力パラメータ	機能コード　10 コンソール・バッファへの読込み	終了時の状態
		コンソールからの入力が バッファに格納

ＣＬ：０ＡＨ
ＤＸ：バッファのオフセット

コンソール・バッファへの読み込み機能は，コンソールからの編集ずみ入力をレジスタDXに格納されているバッファ番地に読み込む．コンソールからの入力は入力バッファがオーバーフローした時か，キャリッジ・リターン（コントロール-M）またはライン・フィード（コントロール-J）を入力した時に終了する．DXによって示される入力バッファは，次の形式をしている．

レジスタDX：	＋0	＋1	＋2	＋3	＋4	＋5	＋6	＋7	＋8	・　・　・	＋n
	mx	nc	c1	c2	c3	c4	c5	c6	c7		??

ここでmxは入力バッファに保持できる最大文字数で，ncは入力バッファに読み込みを行なった文字数である．mxは機能コード10が呼ばれる以前に決まっており，その範囲は1から255の値である．mxに0をセットしても，mxに1をセットしたことと同じになる．ncは0～mxの範囲である．もし，nc<mxのときには，コンソールからの入力文字の後に上図では??で示されているように何が入っているかわからない．入力を編集する種々のコンソール機能は表4-3のとおりである．

表4－3

rub／del　　：	最新の文字を消す．
コントロール-C：	行の最初にシステムの読み込みを再度実行する．
コントロール-E：	行の終わり．
コントロール-H：	1字分バック・スペース．
コントロール-J：	（ライン・フィード（改行））入力ラインの終了．
コントロール-M：	（リターン（復帰））入力ラインの終了．
コントロール-R：	行を替えた後，最新の行を再びタイプイン．
コントロール-U：	行を替えた後，最新の行を取り除く．
コントロール-X：	現在の行の最初までにバック・スペースを繰り返す．

行の左端へ復帰するコントロール-Xのような機能は，プロンプトのカラム位置へ復帰する．これは，データ入力とより読みやすい行修正をオペレータに促す．

入力パラメータ　CL：0BH
機能コード　11
コンソールの状態信号の取り出し
終了時の状態　AL：コンソール・ステータス

コンソール・ステータスの取り出し機能は，論理コンソール装置（CONSOLE）で文字がタイプイン入力されたかどうか調べる．文字がタイプイン入力されたときは，レジスタALには01Hの値が入るが，タイプイン入力がされていないときは，レジスタALには00Hの値が入っている．

入力パラメータ　CL：0CH
機能コード　12
バージョン番号の取り出し
終了時の状態　BX：バージョン番号

バージョン番号の取り出し機能は，バージョンに関係なくプログラムを作成するのに必要なデータを取り出す．レジスタBXに格納される2バイトの値は，次の通りである．

CP／Mオペレーティング・システムの時には，レジスタBH＝00，MP／Mオペレーティング・システムの時にはレジスタBH＝01となる．また，バージョン番号が2.0より前のものは，レジスタBL＝00となる．CP／M 2.0の時には，レジスタBL＝20（16進），それ以降のバージョンでは21，22から2F（16進）となる．CP／M-86オペレーティング・システムの最初のバージョンは，バージョン番号の互換性をもたせるためにCP／M 2.2と同じ値にする．

4.3 BDOSのファイル操作

機能コード13から 52 はCP／M-86オペレーティング・システムでのディスク・ファイルの操作に関するものである．これらの操作の多くでは，レジスタDXにファイル・コントロール，ブロック（FCB）のレジスタDSに対する相対オフセットを格納する．ファイル・コントロール・ブロック（FCB）のデータ領域はシーケンシャル・ファイルのときには連続した33バイト，ファイルをランダムに読み書きするときには36バイトのデータで構成されている．

標準のディフォルト・ファイル・コントロール・ブロックは，通常DSレジスタからオフセット005CHに位置する．従って，ランダム・ファイル・アクセスの場合には，以下に説明する様にレコード番号のフィールとして007DH，007EH，007FH，の3バイトを使用している．

つぎにファイル・コントロール・ブロック（FCB）の様式をフィールド単位で示す．

dr	f1	f2	／	／	f8	t1	t2	t3	ex	s1	s2	rc	d0	／	／	dn	cr	r0	r1	r2
00	01	02	・	・・	08	09	10	11	12	13	14	15	16	・	・・	31	32	33	34	35

ここで，

dr　ディスク・ドライブ・コード（0～16）

0→ファイルを標準ディスク・ドライブに割当て

1→自動ディスク選択して，ディスク・ドライブAを選択

2→自動ディスク選択してディスク・ドライブBを選択

⋮

16→自動ディスク選択して，ディスク・ドライブPを選択

f1……f8

ASCIIコードでファイル名を格納．

ASCIIコードとしてはアッパーケースを選択して，ハイ・ビッドをゼロにしたものを使用する．

t1, t2, t3

ASCIIコードでファイルの型を格納．

ASCIIコードとしてはアッパーケース を選択して，ハイ・ビッドをゼロにしたものを使用する．t1'，t2' および t3' でこれらのビットの値を表わしたとき，

t1'＝1→読み出しのみ可能なファイル

t2'＝1→システム・ファイル，DIRリストなし

ex　論理的領域（ロジカル・エクステント）の現在位置を示す番号で，通常ユーザによって00にセットされるが，ファイルが入出力を行なっているときには0～31の範囲の値をとる．

s1　システム内部で使用している．

s2　システム内部で使用している．OPEN，MAKEおよびSEARCHを呼び出すときにはゼロにセットする．

rc　〝ex〟で示された論理的領域（ロジカル・エクステント）のレコード番号で0から128の範囲の値をとる．

d0……dn

CP／Mシステムによってセットする値で，システムで使用している．

cr　シーケンシャル・ファイルの処理を行なっているときに，読み出しあるいは書き込みを行なっているレコード位置を示す値で，通常ユーザによってゼロにセットされる．

r0, r1, r2

ランダム・レコードのレコード番号で，0から65535の範囲の値をとり，r0，r1で16ビットを構成し，r0が低位のけたで，r1が高位のけた，r2がオーバーフローとなっている．

古いバージョンのCP／Mオペレーティング・システム利用ユーザのために，CP／Mバージョン２とCP／M-86ではディレクトリを使用する処理は，書き込みバッファに影響を与えない予約領域で行なわれることを記しておく．ただし，最初のデータのサーチおよび次のデータのサーチ機能の場合は，ディレクトリ・レコードが現在のDMAアドレスへコピーされるので除かれる．

ファイルの処理中にBDOSが出合うエラー状況は３つある．その内の１つが発生するとBDOSは次のエラー・メッセージをコンソールへ出す．

BDOS　ERR　ON　x：error

ここでxは，エラーが発見された時選択されていたドライブのドライブ名〝error〟は以下の３つのエラー・メッセージのうちのどれか１つである．

BAD SECTOR　　SELECT　　R／O

これらのエラー状況をBDOSが検出すると，実行中のトランジェント・プログラムは一時停止する．エラー状況の指示はトランジェント・プログラムへは何も返されない．

〝BAD　SECTOR〟エラーは，BIOSモジュールからBDOSへ返されたエラー条件の結果として起る．BDOSはシステム・コールに関するファイルの実行の一部としてBIOSセクターの読み書きコマンドを実行し，BIOSの読出しや書込みを行なうルーチンがハードウェア上のエラーを検出すると，このエラー・メッセージを出力するようにBDOSへエラー・コードを返す．

このエラーが出たときは２通りの解決方法がある．１つはコントロール-Cをタイプ・インして実行中のプログラムを終了させる．別の方法はCP／M-86ではリターンをタイプするとエラーを無視し，プログラムの続行を許している．

〝SELECT〟エラーもまたBIOSモジュールからBDOSへ返されたエラー条件の結果として生じる．BDOSはBIOSによる特定のドライブへの読み書き操作に先だってBIOSのディスク・セレクト・コールを行なう．そのドライブがBIOSモジュールでサポートされていないと，このエラー・メッセージを出力するようにBDOSへエラー・コードを返す．CP／M-86では現在実行しているプログラムを終了させ，コンソールからの入力を待つCCPのコマンド・レベルへ制御を戻す．

〝R／O〟エラー・メッセージは，ディスケットがリード・オンリー状態にあるとき，ディスケットへ書き込む操作をすると起る．

ディスク・ドライブがリード・オンリーの状態となるには，STATコマンドによる場合か，BDOS機能を呼び出した場合がある．また〝ウォーム・スタート〟を行なわないでディスケットを交換したときにもリード・オンリー状態となる．ディスケットが交換されたか否かはBIOSで見つけることができる．ただし，選択したドライブにチェックサム・ベクトルが含まれているときだけである．このとき，キーボードへ何かタイプ・インするとトランジェント・プログラムは終了し，CCPへ制御が戻る．

入力パラメータ　　機能コード　13　　終了時の状態

CL：0DH　　ディスク・システムのリセット

ディスク・システムのリセット機能は，プログラムによってディスク・システムをリセット状態にする．ディスク・システムのリセット状態では，全てのディスクの読み書きができ（機能コード28および29参照），ディスク・ドライブAが選択されている．この機能はシステムの読込みを再度実行することなく，ディスクの交換を行なうようなアプリケーション・プログラムによって使用される．機能コード37（ドライブのリセット）も，同様の目的に使用できる．

入力パラメータ　｜　機能コード　14　ディスクの選択　｜　終了時の状態

CL：0EH
DL：選択するディスク番号

ディスクの選択機能は，レジスタDLに格納されている選択するディスク番号を，この命令より後のファイル操作における標準ディスクとして指定する．レジスタDL＝00のときにはディスク・ドライブA，1のときはディスク・ドライブB．このようにして15のときにはディスク・ドライブP（システムとして備えることのでき最大数は16ドライブ）を指定する．さらに，指定したディスク・ドライブはリセット状態にあるなら，それはログ・インされている．〝オン・ライン〞の状態におかれているログ・イン　ディスク・ドライブは，つぎにコールド・スタート，ウォーム・スタート，ディスク・システムのリセットあるいはドライブのリセットを行なうまでは，そのディスク・ディレクトリを常に使用可能状態にしている．

ファイル・コントロール・ブロック（FCB）のdrフィールドの値がゼロのときには，現在選択されている標準ディスク・ドライブが参照される．drフィールドの値が1から16のときには，現在選択されている標準ディスク・ドライブではなく，直接ディスク・ドライブAからPの指定されたディスク・ドライブが参照される．

入力パラメータ　｜　機能コード　15　ファイルのオープン　｜　終了時の状態

CL：~~05H~~ 0F
DX：ファイル･コントロール･ブロック（FCB)のオフセット

AL：リターン・コード

ファイルのオープン機能は，現在実行中のユーザ番号をもち，ディスク・ディレクトリに存在するファイルを指定しているファイル・コントロール・ブロック（FCB）を利用可能にする．

BDOSはレジスタDXで参照したファイル・コントロール・ブロック（FCB）のバイト・ポジション0で指定したドライブのディスク・ディレクトリを探して，ファイル・コントロール・ブロック（FCB）のバイト・ポジション1から12と比べ同じものを探す．またこのときASCIIコードの疑問符（3FH；?）はどこにあろうとも，ディスク・ディレクトリの中のどのような文字とも同じという扱い方をする．通常ファイルのオープン機能を行なうときには疑問符は含まれず，次のexのバイトはゼロとなっている．

ディスク・ディレクトリの要素が同じだったときには，それに関連したディレクトリのデータをファイル・コントロール・ブロック（FCB）のd0からdnフィールドにコピーし，ファイルのオープン機能の後にでてくるファイルの読出し，書込み機能ができるようにする．しかもファイルのオープンやファイル作成機能によってファイル・コントロール・ブロック(FCB)が使用可能の状態になっていないと，BDOSの読出しや書込み命令で使用できない．ファイルのオープン機能を正常に終了すると，ディレクトリ・コードは0から3のいずれかになる．ファイルが見つからなかったときには，ディレクトリ・コードは0FFH（10進の255）となる．またファイル・コントロール・ブロック（FCB）の疑問符号（?）があるときには，最初に出会ったFCBが利用される．またファイルをシーケンシャルに第1レコードからアクセスするときには，確実にプログラムによりcrフィールドをゼロにしなければならない．

入力パラメータ　｜　機能コード　16　ファイルのクローズ　｜　終了時の状態

CL：10H
DX：ファイル･コントロール･ブロック（FCB)のオフセット

AL：リターン・コード

ファイルのクローズ機能は，ファイルのオープン機能の反対のことを行なう．レジスタDXに格納されているファイル・コントロール・ブロック（FCB）のオフセットで示されているファイルは，機能コード15または22によりすでに利用可能となっているが，ファイルのクローズ機能ではディスク・ディレクトリに最新のファイル・コントロール・ブロック（FCB）の内容を記録する．ファイル・コントロール・ブロック（FCB）の照合方法は，ファイルのオープン機能のときと同様である．

ファイルのクローズ機能を正常に終了したときには，ディレクトリ・コードは0, 1, 2または3のいずれかとなる．ディスク・ディレクトリの中にファイル名が見つからなかったときには，ディレクトリ・コードは０ＦＦＨ（10進の255）となり，またファイルの読出しのみしか行なわれなかったときには，ファイルのクローズは行なう必要はない．しかしながら，書込み処理を行なったときは，ディスク・ディレクトリのデータを更新するためにファイルのクローズ処理が必要となる．

入力パラメータ

機能コード　17
最初のディレクトリのサーチ

終了時の状態

ＣＬ：11H
ＤＸ：ファイル・コントロール・ブロック
（ＦＣＢ）のオフセット

ＡＬ：ディレクトリ・コード

最初のディレクトリのサーチ機能は，レジスタＤＸに格納されているファイル・コントロール・ブロック（ＦＣＢ）の番地で示されているファイルと一致するディレクトリを探す．

ファイルが見つからないときには，終了時のディレクトル・コードは255（16進でＦＦ）となる．ファイルが見つかったときには，終了時のディレクトリ・コードは0, 1, 2または3のいずれかになる．ファイルが見つかったときには，ディレクトリ・エントリを内容とするレコードを現在のＤＭＡアドレスのバッファへ記録する．

また開始番地（相対番地）はレジスタＡＬ＊32（レジスタＡＬを５ビット左へローテイトする）となる．ディレクトリのデータは通常応用プログラムでは利用されないが，このようにバッファから抽出することができる．

ＡＳＣIIコードの疑問符号（10進で63，16進で３Ｆ）はフィールド〝f1〞からフィールド〝ex〞のどこにあろうとも，標準もしくは自動選択されたディスク・ドライブのどのディレクトリ・エントリのどのような文字とも同じ扱いをする．

フィールド〝dr〞がＡＳＣIIコードの疑問符号のときには，ディスクの自動選択機能は使用できなくなり，標準ディスクをサーチして，ユーザ番号の一致するエントリを，メモリに割り当てられているか否かにかかわらず探し出す．この後者の機能は，通常は応用プログラムでは使用しないが，現在持っているディレクトリの値を全て探すときに使用できる．フィールド〝dr〞がＡＳＣIIコード疑問符号でないときは，フィールド〝s2〞は自動的にゼロとなる．

入力パラメータ

機能コード　18
次のディレクトリのサーチ

終了時の状態

ＣＬ：12H

ＡＬ：ディレクトリ・コード

次のデータのサーチ機能は，前回一致したエントリからサーチをする以外は，最初のディレクトリのサーチ機能（機能コード　17）と同じである．最初のデータのサーチ機能と同様に，これ以上一致するものがないときには，次のデータのサーチ機能のときにも，ディレクトリ・コードを255（10進）とする．実行の順序では，機能コード18は機能コード17または18の次に呼び出さなければならない．さらに，他のディスクに関連するＢＤＯＳ機能をその間に呼び出してはいけない．

入力パラメータ

機能コード　19
ファイルの削除

終了時のパラメータ

ＣＬ：13H
ＤＸ：ファイル・コントロール・ブロック
（ＦＣＢ）のオフセット

ＡＬ：リターン・コード

ファイルの削除機能は，レジスタＤＸに格納されているファイル・コントロール・ブロック（ＦＣＢ）と一致しているファイルを削除する．ファイル名およびファイルの形には疑問符号を含んでいてもかまわないが，ドライブの選択コードには疑問符号を含むことができないのは，最初のデータのサーチ機能や次のデータのサーチ機能のときと同じである．

ファイルの削除機能においても，削除しようとしているファイルが見つからないときには，ディレクトリ・コードは０ＦＦＨ（10進で255）となり，ファイルが見つかり削除を行なったときには，リターン・コードは０になる．

入力パラメータ	機能コード　20 シーケンシャルな読出し	終了時の状態
CL：14H DX：ファイル·コントロール·ブロック （FCB）のオフセット		AL：リターン・コード

この機能を実行するときには，レジスタDXに格納されているファイル・コントロール・ブロック（FCB）の番地で示されるファイルは，ファイルのオープン機能（機能コード　15）またはファイルの作成機能（機能コード　22）により使用可能な状態にされていなければならない．シーケンシャルな読出し機能は，ファイルからシーケンシャルに128バイト読み，現在のDMAアドレスで指定されているメモリ番地に格納する．

読み出されるレコードの論理領域（ロジカル・エクステント）に占る位置は，フィールド〝cr〟がオーバ・フローするときには，つぎのレコードを示すために，つぎの論理領域（ロジカル・エクステント）をオープンして，フィールド〝cr〟をゼロにリセットする．シーケンシャルにファイルの第1レコードから読み出すなら，ファイルのオープンの次にフィールド〝cr〟をゼロにセットする必要がある．読出し動作が正常に行なわれたときには，リターン・コードを00Hとし，リターン・コードが01Hのときは，つぎのレコード位置にデータがなかった（エンド・オブ・ファイルとなったときなど）ことを表わす．通常はエンド・オブ・ファイルに出会った場合である．しかしながら，前もって書き込みが行なわれなかったデータ・ブロックや生成されていない論理領域（ロジカル・エクステント）の読出し動作の場合もある．このような状況はBDOSのランダムな書込みコマンド（機能コード　34）によってファイルを作成したり追加したときである．

入力パラメータ	機能コード　21 シーケンシャルな書込み	終了時の状態
CL：15H DX：ファイル·コントロール·ブロック （FCB）のオフセット		AL：リターン・コード

この機能を実行するときには，レジスタDXに格納されているファイル・コントロール・ブロック（FCB）の番地で示されるファイルは，ファイルのオープン機能（機能コード　15）またはファイルの作成機能（機能コード　22）により，使用可能な状態にされていなければならない．

シーケンシャルな書込み機能は，ファイル・コントロール・ブロック（FCB）の番地で示されるファイルへ現在のDMAアドレスで示される128バイトのデータ・レコードを書き込む．書き込まれるレコードは，フィールド〝cr〟によって示されている．また書き込み実行後，自動的にフィールド〝cr〟の値は，つぎのレコード位置を示すように更新される．

フィールド〝cr〟がオーバーフローするときは，つぎのレコードを示すために，つぎの論理領域（ロジカル・エクステント）をオープンして，フィールド〝cr〟をゼロにリセットする．シーケンシャルな書き込み機能はすでにデータが格納されているファイルに対しても実行できる．すでにデータが格納されているファイルに対して書き込みを行なったときには，前のデータは消えてしまう．シーケンシャルにファイルの第1レコードから書き込むなら，ファイルのオープンや作成の後でフィールド〝cr〟にゼロをセットする必要がある．

書込み動作が正常に行なわれたときには，リターン・コードを00Hとする．リターン・コードがゼロでないときには，つぎの条件によって，書き込みが正常に行なわれなかったことを表わす．

01：ディレクトリ領域のスペースがない．

　　選択したディスク・ドライブに，新しいディレクトリ・エントリに対する論理的領域（ロジカル・エクステント）を生成しようとしたが，ディレクトリ領域がもはや存在しないときに起こる．

02：データ・ブロック領域のスペースがない．

　　選択したディスク・ドライブのファイルに，新しいデータ・ブロックをアロケートしようとしたが，もはや，データ・ブロックが存在しないときに起こる．

入力パラメータ

機能コード　22
ファイルの作成

終了時の状態

CL：16H
DX：ファイル・コントロール・ブロック
　　（FCB）のオフセット

AL：リターン・コード

ファイルの作成機能は，現在ディスク・ディレクトリに存在しない（すなわちフィールド〝dr〟が明らかにゼロでないかまたは，フィールド〝dr〟がゼロのときには標準ディスク）ファイルを使用しなければならないこと以外は，ファイルのオープン機能（機能コード　15）と同じである．

ファイルの作成機能を行なっとBDOSはファイルを作成し，かつディレクトリとメモリの初期設定を行ない，データの入っていない空のファイルを作る．プログラムを作成するときには，すでに作成しているものと同じファイル名を使用しないよう注意する必要がある．同じファイル名を使用するときには，必ず前のものを削除してから使用する．

ファイルの作成が正常に行なわれたときには，リターン・コード（レジスタAL）は0, 1, 2または3のどれかになる．ディレクトリ・コードが0FFH（10進で255）のときには，ディレクトリを格納する余裕がなくファイルの作成が正常に行なわれなかったことを表わす．

ファイルの作成機能を行なうと，ファイル・コントロール・ブロック（FCB）は使用可能な状態になっているので，それに続くファイルのオープン機能は省略することができる．

入力パラメータ

機能コード　23
ファイル名のつけ直し

終了時の状態

CL：17H
DX：ファイル・コントロール・ブロック
　　（FCB）のオフセット

AL：リターン・コード

ファイル名のつけ直し機能は，レジスタDXに格納されているファイル・コントロール・ブロック（FCB）のオフセットを用いて，最初の16バイトの名前を持つファイルを第2番目の16バイトを持つファイル名につけ直す．指定したファイル名が正しいCP／Mファイル名であることを保証するのはユーザの責任である．

フィールド〝dr〟のディスク・ドライブ・コードは，名前をつけ直したいファイルの存在するドライブを指定する．ファイル名のつけ直しが正常に行なわれたときには，リターン・コード（レジスタAL）は0，である．最初のファイル名が，ディレクトリの中になかったときには，リターン・コードは0FFH（10進で255）となり，ファイル名のつけ直しが正常に行なわれなかったことを示す．

入力パラメータ

機能コード　24
ログイン・ベクトルの取出し

終了時の状態

CL：18H
BX：ログイン・ベクトル

BX：ログイン・ベクトル

ログイン・ベクトルの取出し機能は，CP／M-86オペレーティング・システムの働きによって，16ビットのログイン・ベクトルをレジスタBXに取出す．ログイン・ベクトルの16ビットは，最下位ビットが第1番目のドライブAに，最上位ビットが第16番目のドライブPに対応しており，ビットの値が0のときには，そのドライブがオンラインで使用されていないが，ビットの値が1のときには，選択しているディスク・ドライブがオンラインで使用されているかあるいはフィールド〝dr〟の値がゼロでないときに，ファイル処理機能によって選択されたディスク・ドライブがオンラインで使用されていることを示している．

入力パラメータ ／ 機能コード 25 ／ 終了時の状態

選択されているディスク番号の取出し

CL：19H

AL：選択されているディスク番号

選択されているディスク番号の取出し機能は，レジスタALに現在選択されている標準ディスク番号を取り出す．ディスク番号の値の範囲は，0から15までで，それぞれディスク・ドライブAからディスク・ドライブPに対応している．

入力パラメータ ／ 機能コード 26 ／ 終了時の状態

DMAアドレスのセット

CL：1AH
DX：DMAオフセット

DMAはメモリの絶対番地（Direct Memory Address）の略．ディスク・サブシステムとコンピュータとのメモリの間のデータの送受を行なっているディスク・コントロールとの関連で，DMAアドレスはしばしば使用される．DMAアドレスを使用しないで，I／O機能をプログラムによってコントロールしているコンピュータ・システムもいろいろあるが，CP／Mオペレーティング・システムではディスクへの書込みおよびディスクからの読出しを，128バイトのレコード単位で行なうのにDMAアドレスが使用される．

CP／M-86オペレーティング・システムでは，DMAアドレスのセット機能は，現在のDMAベース・アドレスからの入出力用バッファのオフセットを指定するために使用される．それ故，DMAアドレスの指定には機能コード 26と機能コード51が呼ばれる．このようにDMAアドレスは，レジスタDXに格納されているDMAオフセット＋DMAベースで指定された値となる．このDMAアドレスは次のDMAアドレスのセット機能を行なうか，DMAベースのセット機能を行なうまで変らない．

入力パラメータ ／ 機能コード 27 ／ 終了時の状態

アロケーション・アドレスの取出し

CL：1BH

BX：アロケーション・オフセット
ES：セグメント・ベース

メモリには，各オンラインのディスクに対応してアロケーション・ベクトルが保持されている．アロケーション・ベクトルに保持されているデータを使用して，各種のシステム・プログラムはまだ使用可能な記憶容量の大きさを調べることができる．（STATプログラム参照）．

アロケーション・アドレスの取出し機能は，現在選択されているディスク・ドライブのアロケーション・ベクトルのセグメント・ベースとオフセット・アドレスを取出して，レジスタESとレジスタBXに格納する．ただし，選択されているディスク・ドライブが読出しのみ可能なファイルに指定されているときには，このアロケーション・ベクトルの内容は異なる意味をもってくる．

入力パラメータ ／ 機能コード 28 ／ 終了時の状態

書き込み保護ディスク

CL：1CH

書き込み保護ディスク機能は，現在選択されているディスクに一時的な書き込み保護を指定する．この指定を行なうと，次にコールド・スタート，ウォーム・スタート，ディスクのリセット，またはドライブのリセットを行なうまでは，ディスクに書込みを行なおうとすると，つぎのメッセージを表示する．

BDOS　ERR　ON　d：R／O

入力パラメータ	機能コード　29 R／O(読み出しのみ可能) ベクトルの取出し	終了時の状態
CL：1DH		BX：R／Oベクトル

R／Oベクトルの取り出し機能は，一時的に読出しのみ可能なディスクに指定されているディスク・ドライブを示すR／Oベクトルをレジスタ BXにとり出す．機能コード24と同様に，R／Oベクトルの16ビットは，最下位ビットが第1番目のドライブAに，最上位ビットが第16番目のドライブPに対応しており，それぞれに対応するディスクが機能コード28（書込み保護ディスク）またはCM／P-86 オペレーティング・システムによって書込み保護にセットされているときに各ビットはセットされる．

入力パラメータ	機能コード　30 ファイル属性のセット	終了時の状態
CL：1EH DX：FCBオフセット		AL：リターン・コード

ファイル属性のセット機能は，ファイルに付けられた固定の属性をプログラムにより取り扱えるように，既ち読出し可能なファイル指定およびシステムの属性（t1'，t2'およびt3'）をセットしたり，リセットすることができるようにする．レジスタDXに格納されているFCBの番地で指定されたファイルの属性をセットしたり，リセットしたりする．

利用者はファイル名を正確に指定する責任がある．機能コード30は，現在のユーザ・コードに属し，かつFCBで指定したファイル名とタイプ・フィールドが一致するデフォルト・ディスク・ドライブのディレクトリを探し，そのファイルのディレクトリ・エントリを更新する．

f1'からf4'ビットの値は，ファイルのオープン機能やクローズ機能では使用されていないので，現在は使われていないが，応用プログラムでは使用することができる．f5'からf8'ビットの値は，将来に於けるシステム拡張用のものである．指定された属性を次のように定義する．

t1'：R／O属性はもしセットされていれば，ファイルがリード・オンリ状態を表す．BDOSは，R／Oスティタスのファイルに対するライト・コマンドを許さない．

t2'：システム属性はCP／MのDIRコマンドで参照されるが，セットされていてもファイルのディレクトリを表示しない．

t3'：保管属性が予約されているが，CP／M-86では使用していない．もしセットされていると，利用者によってプログラムを保管するためバックアップ・ストーレッジ用としてファイルが作成されていることを示す．
この機能を実行するには，ファイルをバックアップ・ストーレッジにコピーするときに，保管プログラムでこの保管属性をセットする．ファイルを更新したり新規に作成するプログラムでは，この保管属性をリセットする．しかし，保管プログラムは保管属性をリセットしたファイルだけをバックアップする．このように，自動バックアップ機能は，ファイルを変更するためには制限があるが容易に実行することができる．

機能コード30は，参照しようとしたファイルがないときはレジスタALに0FFH（10進で255）を格納して戻る．そうでなければゼロの値が返る．

入力パラメータ	機能コード　31 ディスク・パラメータ・ アドレスの取出し	終了時の状態
CL：1FH		BX：ディスク・パラメータ・ 　　ブロックのオフセット ES：セグメント・ベース

ディスク・パラメータ・アドレスの取出し機能は，レジスタBXとESに基本I／Oシステム（BIOS）にあるディスク・パラメータ・ブロックのオフセットとセグメント・ベースを格納する．このアドレスは次の2つの目的のために使われる．1つはディスク・パラメータの値の表示と空き領域の計算のために取り出す．またはディスクの使用条件が変更したときに，トランジェント・プログラムによって現在のディスク・パラメータの値を変更する必要が生じたときに使用する．通常，応用プログラムでは，この機能を使用することはない．

6.3節で，BIOSディスク・パラメータ・ブロックを定義している．

入力パラメータ	機能コード　32 ユーザ・コードのセット または取出し	終了時の状態
CL：20H DL：0FFH（取出し）または 　　ユーザ・コード（セット）		AL：現在のユーザ・コード（取出し） 　　値なし　　　　　　　　（セット）

ユーザ・コードのセットまたは取出し機能を使用して，応用プログラムから現在のユーザ番号を変更したり，質問することができる．レジスタDL＝0FFHのときには，現在のユーザ番号がレジスタALに格納される．ユーザ番号の値は0から15の範囲である．レジスタDLの値が0FFHでないときは，レジスタDLの値（モード16）が現在のユーザ番号となる．

入力パラメータ	機能コード　33 ランダムな読み出し	終了時の状態
CL：21H DX：ファイル・コントロール・ブロック 　　（FCB）のオフセット		AL：リターン・コード

ランダムな読出し機能は，以前のリリースでのシーケンシャル・ファイルの読出し機能とよく以た機能をもっている．ただし，ランダムな読出し機能においては，ファイル・コントロール・ブロックのフィールドr0, r1, r2（バイト33, 34, 35）に格納されている24ビットの値で示されているレコード番号を持つ特定レコードの読出しを行なう．フィールドr0, r1, r2の24ビットの値は，LSBはr0にMSBはr1に，HSBはr2に順に格納されている．CP／Mオペレーティング・システムにおいてはMSBとなっているフィールドr2は，ファイルの大きさの計算（機能コード35）以外のときには参照しない．フィールドr2は通常はゼロとなっている．ゼロでないときは，エンド・オブ・ファイル以上のレコードがあり，オーバーフローしていることを表わしている．

このように，r0, r1，は2バイトの値または2バイトの語のように扱われ，読出しを行なうレコード番号が入っている．この値は0から65535の範囲の値で，各サイズのファイルから特定のレコードをとり出すのに使う．ランダム・アクセス・ファイルを処理するときには，先ず第1番目のエクステントをオープンする．この第1番目のエクステントにデータが割りつけられているか否かにかかわらず，このようにすることによりFCBが適当に初期化されて，直接入出力が行なえるようになる．そして選択したレコード番号をランダム・レコード・フィールドr0, r1に格納した後,レコードを読み出すために基本ディスク・オペレーティング・システム（BDOS）を呼び出す．

BDOSから戻ったときには，ランダムな読み出しが正常に行なわれたときにはリターン・コード（レジスタAL）は00となり，正常に行なわれなかったときにはリターン・コードは下に示すエラーコードのどれか1つが格納されている．ランダムな読出しが正常に行なわれたときには，DMAアドレスのバッファには，現在読み出しを行なったレコードが入る．またシーケンシャルな読出し機能とことなり，レコード番号は，つぎのレコード番号へ進まない．そこで，これに続くランダムな読出しでは，同じレコードを読み出す．

ランダムな読出しを行なうと，ロジカル・エクステントと現在のレコードの値を自動的にセットする．そこでファイルのシーケンシャルな読出しあるいは書込みを，ランダムな読出しあるいは書込みを行なった位置から開始することができる．このようなときには，読出しにおいてはランダムな読出しを行なってからシーケンシャルな読出しを行なうと，最後にランダムな読出しを行なったレコードが，まず最初に読出されてくるし，ランダムな書込みを行なってからシーケンシャルな書込みを行なうと，最後にランダムな書込みを行なったのと同じレコードにまず最初に書込みを行なう．なお，ランダムな読出し，書込みのあとに続いて行なったシーケンシャルな読み出し，書込みによっても，ランダムな読み出しまたは書き込みのためのレコード位置は更新されない．

ランダムな読出し機能を実行したときに，レジスタＡＬに格納されるエラー・コードを表４－４に示す．

表4-4　機能コード　33（ランダムな読出し）のエラー・コード

コード　　　意　味

01：書込まれていないデータの読出し――このエラー・コードはまだ書込まれていないデータ・ブロックをアクセスしようとしたときに発生する．

02：(このランダムな読み出しでは発生しない)

03：現在のエクステントをクローズすることができない．――ＦＣＢのフィールドr0, r1で指定したレコードを含む新しいエクステントへ移る前に，現在のエクステントをクローズすることができなかったとき，このエラー・コードが発生する．
エラーの原因は，ＦＣＢを重複して書込んだか，またはオープンされていないＦＣＢを用いてランダムな読出しを行なったことによる．

04：書込まれていないエクステントへのシーク．――まだ生成されていないエクステントをアクセスしようとした時に発生する．エラー・コード01と同じ状況である．

05：(ランダムな読み出しでは発生しない)

06：ランダム・レコード・ナンバーが範囲外――このエラー・コードはＦＣＢのフィールドr2がゼロでない時つねに発生する．

通常，リターン・コードがゼロ以外のときはデータが正しく読み出されてないことを示し，ゼロのときは正常に読み出されたことを示している．

入力パラメータ

機能コード　34

ランダムな書込み

終了時の状態

ＣＬ：22Ｈ
ＤＸ：ファイル・コントロール・ブロック
（ＦＣＢ）のオフセット

ＡＬ：リターン・コード

ランダムな書込み機能は，ＤＭＡアドレスで示されるメモリからディスクにデータが書き込まれること以外は，ランダムな読出し機能と同じように作動する．しかもディスクの書込みをしようとしている論理領域（ロジカル・エクステント）またはデータ・ブロックが割り当てられていないときには，書込み動作に先立って割当てが行なわれる．ランダムな読出し機能のときと同じようにレコード番号の更新は行なわれない．ファイル・コントロール・ブロック（ＦＣＢ）の論理領域（ロジカル・エクステント）番号および現在のレコード番号は，書込みを行なったレコードの位置にセットされる．またランダムな書込み機能に続いてシーケンシャルな読出し，あるいは書込みを行なうことができる．

ランダムな書込みのあとに行なったシーケンシャルな書込みによって，ランダムな読出し，または書込みのためのレコード位置は更新される．またランダム・モードにあるエクステントの最後のレコードの読込みや書込みによる論理領域（ロジカル・エクステント）番号の更新は，シーケンシャルな読出しあるいは書込みのときと違い行なわれない．

ランダムな書込み機能を使用してファイルのアクセスをするためには，先ず第１番目のエクステント（エクステント０）をオープンしなければならない.ランダムな読出し機能と同様にランダムな書込みをする前にＦＣＢをセットしておく．もしファイルが空なら，第１番目のエクステントに対して，ファイル作成機能を行なわねばならない．第１番目のエクステントにデータが割り付けられているか否かにかかわらず，このようにすることによりファイルの情報がディレクトリに格納され，DIR コマンドで見ることができる．

ランダムな書込み機能が終了すると，レジスタＡＬにエラー・コード(表４-５)かゼロを格納する．ゼロのときは正常終了を示している．

表4-5　機能コード　34（ランダムな書出し）のエラー・コード

コード　　表　　味

01：(ランダムな書き出しでは発生しない)

02：データ・ブロック領域のスペースがない――この状態はランダムな書込みコマンドが新しいデータ・ブロックを割付けようとしたが，選択したディスク・ドライブに割付けるデータ・ブロックがないとき起る．

03：現在のエクステントをクローズすることができない．――ＦＣＢのフィールドr0, r1で指定したレコードを含む新しいエクステントへ移る前に，現在のエクステントをクローズすることができなかったとき，このエラー・コードが発生する．

04：(ランダムな書き出しでは発生しない)

05：ディレクトリ・スペースが十分でない．――この状態は書出しコマンドで新しいディレクトリを必要とする新しいエクステントを生成しようとしたが，選択したディスク・ドライブ上に十分なディレクトリ・エントリがないと発生する．

06：ランダム・レコード・ナンバーが範囲を越えている．――ＦＣＢのフィールドr2がゼロの値でないと常にこのエラー・コードが発生する．

入力パラメータ	機能コード　35 ファイルの大きさの計算	終了時の状態
ＣＬ：23H ＤＸ：ＦＣＢオフセット		ＦＣＢのフィールドr0, r1, r2をセット

ファイルの大きさの計算を行なうときには，r0, r1, r2, を備えている（ランダムな取扱いのため）ファイル・コントロール・ブロックの番地をレジスタＤＸに格納しておかなければならない．またファイル・コントロール・ブロックに入っているファイル名は，ディレクトリを探すときに用いる．

ファイルの大きさの計算を行なうと，フィールドr0, r1, r2には，見かけ上のファイルの大きさ，すなわちつぎにエンド・オブ・ファイルとなるレコード番号を格納する．機能コード35により，高位のバイトr2には01が入り，ファイルが最大のレコードを格納しているときにはレコード数 65536 が，その他のときにはファイルの大きさがr0およびr1に16ビットの値として(ＬＳＢ：r0，ＭＳＢ：r1）格納される．

機能コード35を呼び出すと，すでに存在するファイルのエンド・オブ・ファイルのレコード位置にデータを位置づけ，その後ランダムな書込みを行なうと，位置づけたところにデータを書込むことができる．

シーケンシャルな書込みを行なったファイルの大きさは，見かけ上のファイルの大きさと同じであるが，ランダムな書込みにより作られたファイルには，データを書込んでいない場所があるので，ファイルには見かけ上の大きさで示されるよりも少ないデータしか格納されていない．たとえば，レコード番号65535（ＣＰ／Ｍの最大レコード番号）のレコードをランダムに書込んだときには，実際には１ブロックしかデータが格納されていないにもかかわらず，見かけ上のファイルの大きさは65536レコードとなる．

入力パラメータ　|　機能コード　36　ランダム・レコード番号のセット　|　終了時の状態

CL：24H
DX：ファイル・コントロール・ブロック
(FCB)のオフセット

ファイル・コントロール・ブロック(FCB)
のフィールドr0, r1, r2をセット

ランダム・レコード番号のセット機能はシーケンシャルに読出したり，書込みを行なったレコードのランダム・レコード位置に，基本ディスク・オペレーティング・システム（BDOS）が自動的に位置づけ，それに続くランダムな読出しおよび書込みをそこから行なえるようにするが，用法としては次の2つが考えられる．

まず第1は，シーケンシャルにファイルを読み，各種のキー・フィールドの位置を抽出するのに使われる．シーケンシャルに読み，キーが同じものを見つけると，ランダム・レコード番号のセット機能（機能コード　36）を呼び出して，このキーのデータに対応するランダム・レコード番号を計算する．データの大きさが128バイトのときには，後で使用しやすいようにキーとともに，計算したランダム・レコード番号より1小さい値を表に登録する．

シーケンシャルに全ファイルを検索し，キーとランダム・レコード番号の表を作成しておくと，以後は表に書かれているランダム・レコード番号を使って直ちに希望するレコードの読出し，または書込みが行なえる．このような方法は，可変長レコードのときにも容易に行なうことができ，可変長レコードのときには，レコード番号のほかに，そのレコード内でのデータの開始位置をバイト単位で表にしておけば可能となる．

第2のランダム・レコード番号のセット機能の用法は，シーケンシャルな読出しあるいは書込みの後，ランダムな読出しあるいは書込みに変更するときに使用する．ファイルをシーケンシャルに読出しあるいは書込みを行なった後，ランダム・レコード番号のセット機能（機能コード　36）を呼び出し，ランダム・レコード番号をセットし，それから今セットしたレコードの位置からランダムな読出しあるいは書込みを続けるときに使用する．

入力パラメータ　|　機能コード　37　リセット・ドライブ　|　終了時の状態

CL：25H
DX：ドライブ・
ベクトル

AL：00H

リセット・ドライブ機能は，指定したドライブをプログラム的にリセット状態にするために使用される（リセット状態のドライブとは，ログインされていず，リード／ライト状態にあることを意味する）．DXレジスタで渡されたパラメータは，リセットするドライブの16ビット・ベクトルであり，ここで最下位ビットは最初のドライブAへ対応しており，最上位ビットは16番目のドライブPに対応している．ビット値〝1〞は指定したドライブがリセットされるということを意味している．

CP／MとMP／Mの互換性を保つために，リターン・コードとしてゼロを戻す．

入力パラメータ　|　機能コード　40　ランダムな書込み(ゼロ・フイル)　|　終了時の状態

CL：28H
DX：FCBオフセット

AL：リターンコード

ゼロ・フィル動作を伴ったランダムな書込みを行なうこの機能は，レコードが書かれる前に未割当のデータ・ブロックが，ゼロを格納されたレコードで初期化されるということを除いて，ランダムな書込み機能（機能　34）と同じである．この機能がファイルを生成するために使用されると，全てのゼロよりなるランダムな読出し操作によってアクセスされるレコードは，未書込みランダム・レコード番号を識別する．ランダムな書込み機能によって生成されたファイルに割当てられた未書込みランダム・レコードは，初期化されていないデータを含んでいる．

入力パラメータ	機能コード 47 プログラムのチェイン	終了時の状態
CL：2FH DMAバッファ：コマンド・ライン		

プログラムのチェイン機能は，オペレータの介在なしにあるプログラムから次のプログラムへ制御を渡す．この呼び出しでは渡されるパラメータはなく，呼び出しプロセスは，ディフォルトDMAバッファにヌル・バイトにより終了するコマンド・ラインを置かねばならない．

CP／M-86のもとでは，この機能はコマンドの実行前に呼び出しプロセスのメモリを解放し，新らしいプログラムのベース・ページを解析し，コマンド・ラインをセットする．それから，コンソール・コマンド・プロセッサ（CCP）が，コマンド・ラインを実行する．

入力パラメータ	機能コード 49 SYSDATアドレスの獲得	終了時の状態
CL：31H		BX：SYSDATアドレス・オフセット ES：SYSDATアドレス・セグメント

ＳＹＳＤＡＴアドレス獲得機能は，システム・データ・エリアのアドレスを戻す．システム・データ・エリアには，次の情報が格納されている．

```
dmaad           equ     word ptr 0      ;user DMA address
dmabase         equ     word ptr 2      ;user DMA base
curdsk          equ     byte ptr 4      ;current user disk
usrcode         equ     byte ptr 5      ;current user number
listcp          equ     byte ptr 22     ;listing toggle...
                                        ;set by ctrl-p
curdrvs         equ     byte ptr 23     ;current drives to
                                        ;search
console_width   equ     byte ptr 64
printer_width   equ     byte ptr 65
console_column  equ     byte ptr 66
printer_column  equ     byte ptr 67
```

入力パラメータ　　機能コード　50　　終了時の状態
ダイレクトBIOSコール
CL：32H
DX：BIOS記述子

機能コード　50は，ダイレクトにBIOS　を呼び出し，BDOSを通してBIOSへ制御を移す．DXレジスタはBIOSコール・パラメータの有る5バイトのメモリ領域のアドレスである．

8ビット	16ビット	16ビット
FUNC	Value（CX）	Value（DX）

ここで，FUNCはBIOS機能番号（表5-1参照）であり，値（CX）と値（DX）はBIOSコールで直接CXとDXレジスタに渡される16ビット値である．CXとDX値はBIOSコールが初期化される以前に8086レジスタへロードされる．

入力パラメータ　　機能コード　51　　終了時の状態
DMAセグメントのベースセット
CL：33H
DX：ベース・アドレス

機能コード　51は，引き続いて起るDMAの転送のためにベース・レジスタをセットする．DXのワード・パラメータは，パラグラフ・アドレスであり，ディスクのリード，ライト機能で使用される128バイト・バッファのアドレスを指定するためにDMAオフセットとともに使用される．イニシャル・プログラ・ローディング時には，ディフォルトDMAベースはユーザのデータ・セグメント（DSの初期値）へ，DMAオフセットは0080Hへセットされており，ベース・ページのディフォルト・バッファをアクセスするようになっていることに注意．

入口　　機能コード　52　　出口
DMAベース・セグメントの獲得
CL：34H
BX：DMAオフセット
ES：DMAセグメント

機能コード　52は，現在のDMAベース・セグメント・アドレスをESへ，オフセットをDXへ格納する．

4.4　ＢＤＯＳメモリ管理とロード

ＣＰ／Ｍ-86の下では，メモリは２通りの異なる方法で割付けられる．第一は，ホスト・システムで利用できる物理的なメモリを定義する静的なアロケーション・マップ（ＢＩＯＳ内にあり）を通してである．この方法では，ＲＡＭかＲＯＭの非連続な８つまでの領域よりなるメモリ構成でＣＰ／Ｍ-86を動かすことが可能である．連続するメモリよりなる単一ＲＡＭベース・システムに於いて，静的なマップは，通常ＢＩＯＳの終りから開始して利用できるメモリの終りまでよりなる１つの領域を定義する．

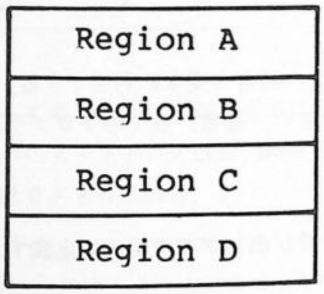
ひとたび，メモリがこの方法で物理的にマップされると，ＣＰ／Ｍ-86は，トランジェント・プログラムのローディングと実行を行うために第２レベルの動的割当を行う．ＣＰ／Ｍ-86は８つの領域にメモリの動的割当を行なう．

割当の依頼は，プログラム・ロード・オペレーションを通して暗黙的に，またはこの章で説明するＢＤＯＳコールを通して明示的に行なわれる．プログラム自身は２通りの方法で，即ちＣＣＰレベルで入力されたコマンドを通じてか，ＢＤＯＳプログラム・ロード・オペレーション（機能コード　59）を通じてロードされる．

マルチプル・プログラムは，各プログラムがシステム・リセット（機能コード０）を実行して，メモリに残っている（ＤＬ＝01Ｈ）限り，ＣＣＰレベルでロードすることができる．このタイプのマルチプル・プログラムだけが割込みを防害することによって制御を受け，そして通常の環境下では一時にメモリに存在するのは，一つのトランジェント・プログラムのみである．しかし，マルチプル・プログラムがメモリに存在するとき，オペレータが入力した，ＣＯＮＴＲＯＬ-Ｃ，たとえどのプログラムがアクティブティにコンソールをリーディングしていても，プログラムがロードされたのと逆の順番でこれらのプログラムを削除する．

ＣＣＰコマンドを通じてロードされたプログラムは，付加的なプログラムをロードしデータ領域を割付けることができる．今メモリの４つの領域に次の順で割付けられるとする．即ち，プログラムがオペレータ・コマンドを通してＣＣＰレベルでロードされ，ＣＭＤファイル・ヘッダーが読まれ，そのプログラムとそのデータより成る全メモリ・イメージが領域Ａへロードされ，実行を開始する．このプログラムは順に，領域Ｂへ別のプログラムをロードするためにＢＤＯＳプログラム・ロード機能（59）を呼び出し，そのロードしたプログラムへ制御を渡す．領域Ｂのプログラムは領域Ｄが続く領域Ｃを割当てる．割付けの順次を下の図4-1に示す．

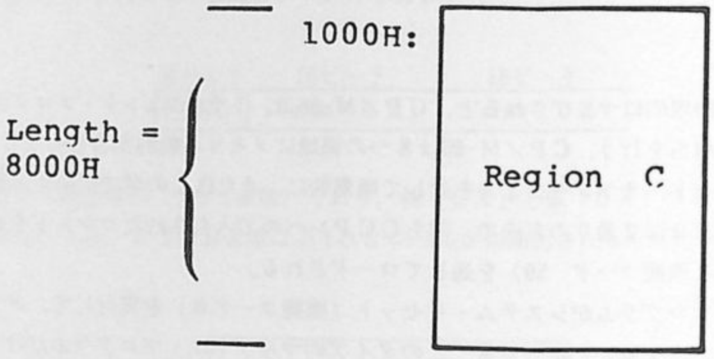
図4-1　メモリ割当の例



これ等の領域には，階層的な所有権がある．即ち，Ａに於けるプログラムはＡからＤまでの全てのメモリを制御する．Ｂにおけるプログラムは領域ＢからＤまでを制御する．Ａにおけるプログラムは，希望するなら領域ＢからＤまでを解放でき，また別のプログラムを再ロードすることもできる．例へば，ＤＤＴ-86はこの方法で別のテスト・プログラムをローディングする前に，現在プログラムが使用しているメモリを解放するため，メモリ領域の解放（機能コード　57）を実行する．さらに，Ｂにおけるプログラムはアプリケーションによって必要ならば領域ＣとＤを解放できる．しかし，ＡかＢのどちらかがシステム・リセット（ＤＬ＝00Ｈを持つＢＤＯＳ機能コード　0）によって終了するときには，ＡからＤの４つの領域の全てが解放されるという事に注意しなければならない．

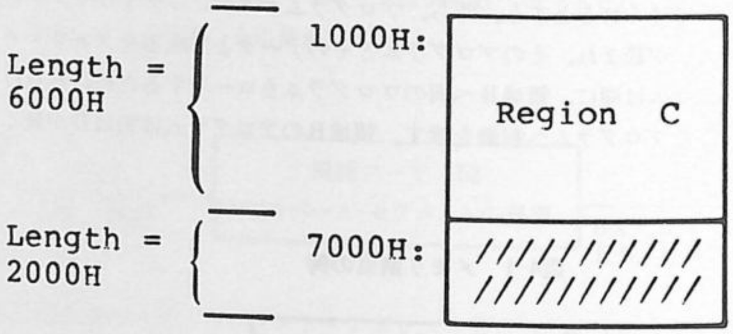
トランジェント・プログラムは領域の1部を解放することが可能であり，次の割当て要求で解放した部分を割付けることもできる．しかし，解放される部分は，領域の先頭か終端でなければならない．例えば，領域Bのプログラムが，次の図4-2で示す様な最初の割当て要求に続いて，パラグラフ・ロケーション100Hから800Hパラグラフを受けたとする．

図4-2　メモリ領域の例



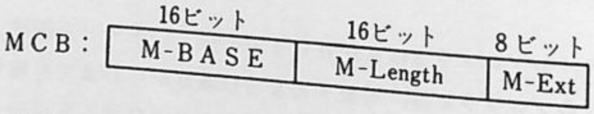
さらに領域Dが割当てられているとする．パラグラフ・ベース700Hから始まる200Hパラグラフを解放することによって，領域Dに影響をおよぼすことなく領域Cの最後の200Hパラグラフが返される．その結果，メモリ配置は図4-3に示す様になる．

図4-3　メモリ領域の例



パラグラフ・アドレス700Hから始まる領域は次の割当て要求に利用できる．メモリ要求は，もしも8つのメモリ領域がすでに割当てられていると，失敗することに注意しなさい．通常，全てのプログラム・ユニットが連続する領域に駐在できるならば，システムはただ1つの領域を割当てる．

機能コード　53から始まるメモリ管理機能は，呼び出しプログラムで定義するメモリ・コントロール・ブロック（MCB）を参照する．MCBは次のような形式である．



ここでM-BASEとM-Lengthは，16バイト・パラグラフ・ユニットで表現されるインプットかアウトプット値のどちらかであり，M-Extは各機能コード毎に特別に定義してあるリターン・バイト値である．エラーはCP／Mのファイル・エラーと一致する様に通常0FFHを返す．

入力パラメータ	機能コード　53 利用可能最大メモリの獲得	終了時の状態
CL：35H DX：MCBの オフセット		AL：リターン・コード

機能コード　53は，M-Lengthパラグラフより小さいか等しい範囲で利用できる最も大きいメモリ領域を見つける．メモリの獲得が成功するとM-Baseには利用できる領域のベース・パラグラフ・アドレス，M-Lengthにはパラグラフ長がセットされる．利用できるメモリがないとALには０FFHが，要求が成功すると00Hが返される．M-Ext は割当ての為に追加メモリがあるならば１，追加のメモリが利用できないときには０がセットされる．

入力パラメータ	機能コード　54 絶対ロケーションにおける 最大メモリの獲得	終了時の状態
CL：36H DX：MCBの オフセット		AL：リターン・コード

機能コード　54はM-Lengthパラグラフの最大値に対し，M-Baseで与えられた絶対パラグラフ境界から最も大きい可能領域を見つけるために使用される．もし成功すれば，M-Lengthは実際の値がセットされる．絶対番地で利用できるメモリがない時には，ALには０FFHが，要求が満されると00Hが返される．

入力パラメータ	機能コード　55 メモリ領域の獲得	終了時の状態
CL：37H DX：MCBの オフセット		AL：リターン・コード

メモリ割当て機能は，DXによって番地づけられたMCBに従ってメモリ領域を割当てる．割当て要求の大きさは，M-Lengthから得られる．機能コード　55は割当てた領域のベース・パラグラフ・アドレスをユーザのMCBに返す．レジスタALには要求が成功したときには00Hが，不成功の場合には０FFHが入れられる．

入力パラメータ	機能コード　56 絶対メモリ領域の獲得	終了時の状態
CL：38H DX：MCBの オフセット		リターン・コード

絶対メモリ領域の割当て機能は，DXによって番地づけられたMCBに従ってメモリ領域を割当てる．割当て要求の大きさは，M-Lengthから得られアブソリュート・ベース・アドレスはM-Baseから得られる．レジスタALには要求が成功したときには00Hが，割当てるメモリがないときには０FFHが入る．

入力パラメータ	機能コード　57 メモリ領域の解放	終了時の状態
CL：39H DX：MCBの オフセット		

機能コード　57は，プログラムへ割当てられたメモリ領域解放するために使用される．M-Ext フィールドの値が，この機能のオペレーションを制御する．即ち，M-Ext＝０FFHの時には呼び出しプログラムによって割当てられた全てのメモリ領域が解放される．また，レジスタDXによって番地づけられるMCBのロケーションM-BaseからM-Length長のメモリ領域を解放する（このときのM-Extフィールドは00Hである）．いずれにおいても，CP／M-86のもとでは割当てられた全領域か領域の終りが解放され，中間領域が解放されることはない．

入力パラメータ	機能コード　58 全メモリの解放	終了時の状態
CL：3AH		

機能コード　58は，CP／M-86環境下の全てのメモリを解放する（通常，初期化に際しCCPだけが使用する）ために使用される．

入力パラメータ	機能コード　59 プログラムのロード	終了時の状態
CL：3BH DX：FCBの 　　　オフセット		AX：リターン・コード／ 　　ベース・ページ・アドレス BX：ベース・ページ・アドレス

機能コード　59は，CMDファイルをロードする．エントリーでは，入力CMDファイルを指しているオープンが成功したFCBのDS相対オフセットがDXレジスタにある．

プログラム・ロードがうまくいかなかったとき，AXには値0FFFFHが入る．一方，ロードが成功した場合には，AXとBXには，ロードしたプログラムに属するベース・ページのパラグラフ・アドレスが入る．ベース・アドレスと各セグメントのセグメント長が，ベース・ページに貯へられる．CCPレベルでプログラムがロードされるとき，DMAベース・アドレスはロードされたプログラムのベース・ページにイニシャライズされ，DMAオフセット・アドレスは0080Hへイニシャライズされるということに注意．しかし，これはCCPの機能であり，機能コード　59はディフォルトDMAアドレスを確立しない．ロードされたプログラムへ制御を渡す前にDMAベースをセットするための機能コード　51，DMAオフセットをセットするための機能コード　26を実行することは，機能コード　59を実行するプログラムの責任である．

第５章
基本Ｉ／Ｏシステム（ＢＩＯＳ）の構成

ＣＰ／Ｍ-86のこのバージョンは，インテル社製のＳＢＣ86／12マイクロコンピューターで動作するように作成されている．しかし，いかなるハードウェアも基本Ｉ／Ｏシステム（ＢＩＯＳ）と呼ばれるサブルーチン群に依存しており，ＣＰ／Ｍ-86システム導入者は，これから述べる方法で8086，8088の動作環境にふさわしいＣＰ／Ｍ-86にするために，これらのサブルーチン群を修飾することができる．

この章では，ＢＩＯＳの各エントリーポイントでの動作を述べ，また，ＢＩＯＳ内で参照される変数とテーブルを定義する．しかし，ディスク定義テーブルについては，この章とは別にこのマニュアルの次章で扱う．

5.1　ＢＩＯＳの構成

ＣＰ／Ｍ-86のＢＩＯＳの部分は，オペレーティング・システムの最も高位部分に位置している．図5-1に一般形式を示す．

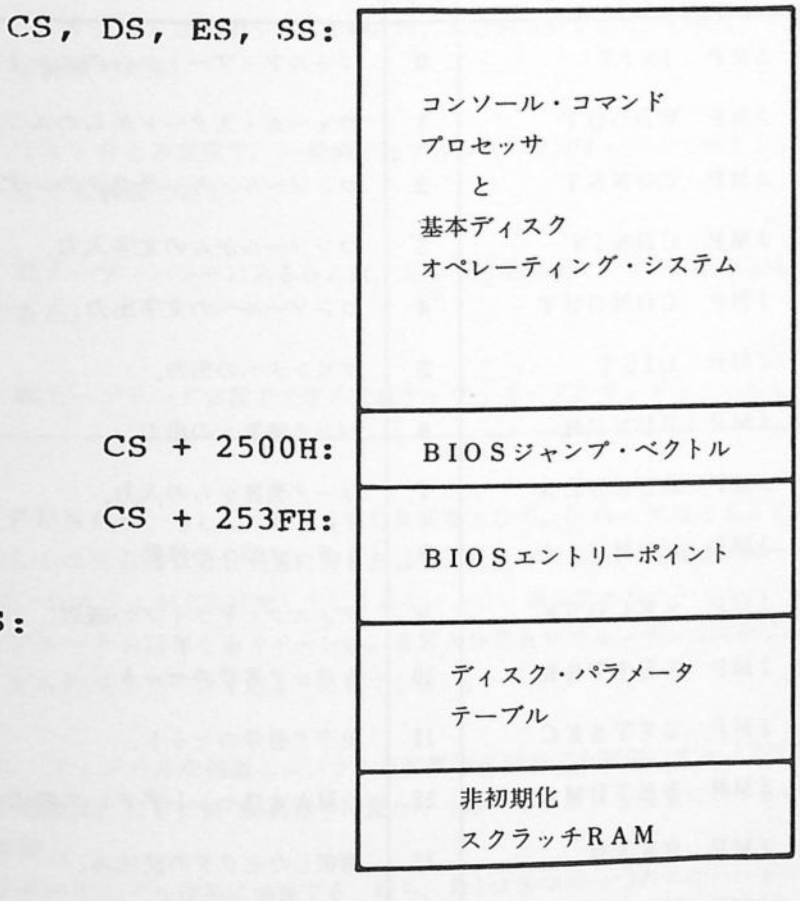


図5-1　ＣＰ／Ｍ-86の一般構成

次章で述べるが，ＣＰ／Ｍ-86ではＣＣＰとＢＤＯＳはＣＰＭ．Ｈ86という16進ファイル形式で提供されている．非標準のハードウェアでＣＰ／Ｍ-86を導入するためには，これからあげる機能を遂行するＢＩＯＳを作成し，ＣＰＭ．Ｈ86ファイルの最後へ16進ファイルを連結しなければならない．また，ＧＥＮＣＭＤユーティリティを使用するとコールド・スタート・ローダがロードできるＣＰＭ．ＳＹＳファイルを作ることができる．メモリにＣＰＭ．ＳＹＳファイルをロードするコールド・スタート・ローダはＢＩＯＳの最も簡単な形式であるＬＤＢＩＯＳ（Loader ＢＩＯＳ）というものを含んでいる．コールド・スタート・ローダはＣＰＭ．ＳＹＳヘッダーで定義したロケーションからＣＰＭ．ＳＹＳをメモリにロードする（通常0400Ｈ）．コールド・スタート・ローダとＣＰ／Ｍ-86ローダーの構築と実行についての手順は，あとの章で述べる．

付録Ｄにインテル204・コントローラ・ボードを用いているインテルＳＢＣ86／12システム用の標準ＣＰ／Ｍ-86ＢＩＯＳのリストを載せてある．付録Ｅには，デバイス・ドライバを動かすのに必要な要素を含むＣＢＩＯＳと呼んでいるＢＩＯＳの骨組みの例を示してある．ＢＩＯＳの全構造を決めるために，これらのリスティングをながめる事が望ましい．

5.2 BIOSジャンプ，ベクトル

BIOSへのエントリーはオペレーティング・システムのベースからオフセット2500Hにある“ジャンプ・ベクトル”を通して行なわれる．ジャンプ・ベクトルはプログラムの制御を個々のBIOSエントリーポイントへ渡すための連続する21個の3バイト命令からなっている．BIOSサブルーチンのある機能は，必要なければ，一つのRET（戻り）であってもよい．しかし，必ず対応するジャンプ・ベクトル・エントリーが，表5-1で示される順になければならない．付録Dに標準CP／M-86 BIOSのリスティングを載せておく．

BIOSの個々のサブルーチンに対するパラメータは，必要ならCXとDXレジスタ経由で渡される．CXは最初のパラメータを受け，DXは二番目の引数のために使用される．返される値はタイプによって異なるが，例えばバイト値はALに返され，ワード値（16ビット）はBXに返される．各々パラメータと返される値は，各サブルーチンごとに説明する．

表5-1　BIOSジャンプ・ベクトル

BIOS先頭からのオフセット	命　令	BIOS F#	意　味
2500H	JMP INIT	0	コールド・ブートからのエントリー・ポイント．
2503H	JMP WBOOT	1	ウォーム・スタートからのエントリー・ポイント．
2506H	JMP CONST	2	コンソール・キャラクタのレディ・チェック．
2509H	JMP CONIN	3	コンソールからの文字入力．
250CH	JMP CONOUT	4	コンソールへの文字出力．
250FH	JMP LIST	5	プリンタへの出力．
2512H	JMP PUNCH	6	パンチ装置への出力．
2515H	JMP READER	7	リーダ装置からの入力．
2518H	JMP HOME	8	トラック00への移動．
251BH	JMP SELDSK	9	ディスク・ドライブの選択．
251EH	JMP SETTRK	10	トラック番号のセット．
2521H	JMP SETSEC	11	セクタ番号のセット．
2524H	JMP SETDMA	12	DMAオフセットアドレスのセット．
2527H	JMP READ	13	選択したセクタの読込み．
252AH	JMP WRITE	14	選択したセクターの書込み．
252DH	JMP LISTST	15	リスト・ステータスを返す．
2530H	JMP SECTRAN	16	セクター変換．
2533H	JMP SETDMAB	17	DMAセグメント・アドレスのセット．
2536H	JMP GETSEGB	18	MEM DESCテーブル・オフセットを得る．
2539H	JMP GETIOB	19	I／Oマッピング・バイトを得る．
253CH	JMP SETIOB	20	I／Oマッピング・バイトのセット．

BIOSジャンプ・テーブルは，大きく分類すると，システムの（再）イニシャライズに関するサブルーチン，文字の入出力に関するサブルーチン，ディスク入出力に関するサブルーチンの3つに分けることができる．

5.3 周辺装置

全ての文字入出力操作は，7 bit　ASCII又は，8 bit　JISのコードで行なわれる．

入力装置に対するend-of-file（ファイルの終了）の条件は，ASCIIコードのコントロールZ（1AH）で与えられる．

周辺装置はCP／M-86により〝ロジカル〟(論理的な）装置として見られ，BIOS内で実際の〝フィジカル〟（物理的）な装置に割りあてられる．装置の特性を表5-2に定義する．

表5-2　CP／M-86〝ロジカル〟（論理的な）装置特性

装置名	特　　性
CONSOLE	オペレータと交信を行う主要な対話用コンソール（制御卓）で，CONST，CONIN，CONOUTが使用する．一般には，CONSOLEは，CRT，テレタイプのような装置である．
LIST	主に，リストをとる装置で，一般的にはプリンタ，又はテレタイプのようなハード・コピーをとる装置である．
PUNCH	主に，紙テープ・パンチにあるならば，ふつうは高速紙テープパンチまたは，テレタイプである．
READER	主に，紙テープリーダ装置で光電子式紙テープ・リーダかテレタイプである．

一つの周辺装置をLIST，PUNCH，もしくはREADER装置として，同時に割当てることができることに注意．もし，LIST，PUNCH，もしくはREADER装置に何も割当てられなければ，PIPか他のユーザ・プログラムにより，その装置がアクセスされた時に，システムが立ち往生してしまわないように適当なエラー・メッセージを出すべきである．代りに，PUNCH，LISTサブルーチンは単なるリターンに，READERサブルーチンは即座にend-of-fileを示すようにAレジスタに1AH（ctl-Z）を入れてリターンするようにしてもよい．

システムの自由度を増すのに，フィジカルな処置とロジカルな装置の再割当てを可能にする〝IOBYTE〟機能を操作することができる．IOBYTE機能は，CP／M-86処理中に変わりうるフィジカルな装置に対するロジカルな装置の割当てを設定する（STATコマンド参照）．

IOBYTE機能の定義は以下のインテル標準に準拠する．即ち，BIOS中の一つのロケーションにIOBYTEと呼ぶ1バイトを確保し，ある期間に効力のあるロジカルな装置のフィジカルな装置に対する割当てを定義する．割当ては，以下のようにIOBYTEを各2ビットずつCONSOLE，READER，PUNCH，LISTと呼ぶ4フィールドに分けることにより行なわれる．

IOBYTE　最上位ビット　……　最下位ビット

LIST	PUNCH	READER	CONSOLE
ビット 6,7	ビット 4,5	ビット 2,3	ビット0,1

各フィールド内の値は，0～3の値で各ロジカル装置の割当てを表わす．各フィールドに割当てられる値を表5-3に示す．

表5-3　IOBYTEフィールド定義

CONSOLEフィールド（ビット0，1）
0――コンソールがコンソール・プリンタ装置へ割当られている（TTY：）
1――コンソールがCRT装置へ割当られている（CRT：）
2――バッチ・モード：CONSOLE入力としてREADERを，CONSOLE出力としてLIST装置を使用する（BAT：）
3――ユーザが定義したコンソール装置（UC1：）

READERフィールド（ビット2，3）
0――READERはテレタイプ装置（TTY：）
1――READERは高速リーダ装置（RDR：）
2――ユーザが定義したリーダ＃1（UR1：）
3――ユーザが定義したリーダ＃2（UR2：）

PUNCHフィールド（ビット4，5）
0――PUNCHはテレタイプ装置（TTY：）
1――PUNCHは高速パンチ装置（PUN：）
2――ユーザが定義したパンチ＃1（UP1：）
3――ユーザが定義したパンチ＃2（UP2：）

LISTフィールド（ビット6，7）
0――LISTはテレタイプ装置（TTY：）
1――LISTはCRT装置（CRT：）
2――LISTはラインプリンタ装置（LPT：）である．
3――ユーザが定義したリスト装置（UL1：）

IOBYTEの設定はオプショナルであり，CBIOSの構成にのみ影響する．フィジカル（物理）装置へのアクセスを許すPIPとロジカル（論理）――フィジカル（物理）割当てを行ったり表示するSTATを除いて，CP／M-86ではIOBYTEを使用していない．いずれにしても，IOBYTEの設定は基本的なCBIOSが完全に完成し，テストされるまで省いた方がけんめいである．その後にIOBYTEをシステム機能を増強するために加えればよい．

5.4　BIOSサブルーチン・エントリー・ポイント

各BIOSサブルーチンのエントリーにおける動きについて以下に述べる．

ディスクI／Oは，常に色々なディスク・アクセス・サブルーチンを連続して呼ぶことによって行なわれることに注意すべきである．これらのサブルーチンは，アクセスするディスク番号や，そのディスクのトラック，セクターおよびダイレクト・メモリ・アクセス（DMA）オフセットとセグメント・アドレスをセットアップする．これらのパラメータがセットアップされた後に，実際のI／O動作を行うREADあるいはWRITEを呼ぶ．

あるディスク・ドライブを選択するためにSELDSKが，単に呼ばれることがしばしばあり，次のオペレーションのために他のドライブを選ぶ前に，今選ばれたディスクへの多数の読書き動作が続くということに注意しなければならない．同様にDMAアドレスが変えられる前に，選んだDMAアドレスから読込んだり出力する幾つかのコールが続くDMAセグメント・ベースをセットするコールやDMAオフセットをセットするコールがある．トラックとセクター・サブルーチンはREAD，WRITEオペレーションが遂行される前に常に呼ばれる．

READ，WRITEサブルーチンは，BDOSへエラー状態を返す前に何回かの再試行（10回が標準）を行なうにちがいない．HOMEサブルーチンは，コントローラーの特性に応じてトラック00へのシークを実際にしたり，しなかったりする．重要なことは，トラック00が次のオペレーションのために選ばれているということであり，そしてそれはしばしばパラメータを00としたSETTRKと全く同じように取り扱われる．

表5-4　BIOSサブルーチンの機能

サブルーチン	内　容
INIT	CPM．SYSファイルがメモリに読込まれた後に，CP／M-86ローダによって直接呼ばれる．この手続きは，ブートストラップ・ローダが行なわなかったハードウェアのイニシャライズ，BIOSの変数（IOBYTEを含む）に対する初期値セット，サインオンメッセージの印字，およびBDOSオフセット（0B11H）とベースを指すインタラプト・ベクトルの初期設定を行なう． このルーチンが終了した後に，CCPオフセット（0H）へジャンプする．全てのセグメント・レジスタは，この時点で，オペレーティング・システムのベースが入っているように，初期化されるべきである．
WBOOT	プログラムが，BDOS機能コード0をコールすることによって終了する時はつねに呼ばれる．ハードウェアやソフトウェアの再イニシャライズはここでおこなわれる．このルーチンが終了すると，CCP（06H）のウォーム・スタート・エントリー・ポイントへ直接ジャンプする．
CONST	現在，割当てられているコンソール装置のステータスを調べ，文字の読込みが可能であれば，ALレジスタに0FFHを，不可能なときにはALレジスタに00Hを返す．
CONIN	コンソールからの文字を読込んで，ALレジスタにそれをセットする． コンソールから入力されなければタイプされるまで待ち，リターンする．
CONOUT	CLレジスタからコンソール出力装置へ文字を送る．文字は，最上位パリティ・ビットが0にセットされたASCIIコードである．もしコンソール装置が行の終りに，時間待ちを必要とするならば（例えば，TIサイレント700ターミナル），ラインフィードかキャリジリターンの時に，時間待ちするようにしてもよい．また，必要ならば，コンソール装置におかしな反応を行すコントロール文字を取り除くこともできる．
LIST	CLレジスタから現在割当てられているリスト装置へ文字を送る．
PUNCH	CLレジスタから現在割当てられているパンチ装置へ文字を送る．

READER　現在割当てられているリーダ装置から，ALレジスタへ文字を読込む.
ファイルの終了（end-of=file）は，ASCIIのコントロールZ（1AH）を返すことにより通知される.

HOME　現在選択されているディスクのディスク・ヘッドをトラック00の位置へもどす．コントローラがトラック00を検出する特殊な機能がないと，HOMEのコールをパラメータ0のSETTRKへのコールに変えることができる.

SELDSK　連続して発生するディスク操作のために，CLレジスタにより指定されたディスクドライブの選択を行う．ここで，ドライブAに対してはCLレジスタには0が，ドライブBに対しては1，ドライブPに対しては15などの様に入る.（市販されている標準CP／M-86は，2つのドライブをサポートする）.

各ディスクの選択では，SELDSKは，選択したディスク・パラメータ・ヘッダーのベース・アドレスをBXレジスタへ返す．標準フロッピー・ディスク・ドライブに対しては，そのヘッダーと関連テーブルの内容は変化しない．CBIOSと呼ばれるCP／M-86に含まれているサンプルBIOSは，SELDSK機能を遂行するプログラム・セグメントを含んでいる．存在しないドライブを選択することを試みると，SELDSKはエラー表示としてBX＝0000Hを返す.

SELDSKは，各コールに対してヘッダーアドレス返さなければならないが，入出力（シーク，リード，ライト）が実際に行なわれるまで物理的なディスク選択操作の実行を延期する方がよい．なぜならば，ディスクの選択は，必ずしもその後にディスク・アクセスを伴うとは限らないからである.

SELDSKのエントリーで，まず最初に指定したディスクが選択されているかどうかを決めることが可能である．ドライブが，現に選択されていないと，DLレジスタのビット0（最下位ビット）は0である．この情報は，ディスク定義テーブルをセット・アップする為にディスクから構成情報を読むシステムでは重要である.

SETTRK　CXレジスタは，現在選択されているドライブの続いておこるアクセスのトラック番号を示している．今指定されたトラックへのシークを行うか，リードかライトが実際に行なわれるまで，シークを遅らせるかのどちらかにすることができる.

CXレジスタは，標準フロッピー・ディスク・ドライブに対しては正しいトラック番号に対応する0―76の範囲の値をとり，非標準ディスク・サブシステムに対しては0―65535の値をとる.

SETSEC　CXレジスタは，現在選択されているドライブの続いておこるアクセスの変換されたセクター番号を示している（後述のSECTRAN参照）．今この情報をコントローラに送るか，それともリードかライトの操作が起こるまで，延期するかのどちらかにすることができる.

SETDMA　CXレジスタが，続いて起こるリード，ライト操作のDMA（ダイレクト・メモリ・アクセス）のアドレスを示している．例えば，SETDMAが呼ばれたときに，CX＝80Hであれば，次のSETDMAやSETDMBのコールが起こるまで，全ての続いて起こるリード操作は，データを現DMAセグメント・ベースのオフセット80Hから0FFHまで読込み，ライト操作は，80Hから0FFHまでをディスクに書込む.

ただし，コントローラは実際には，ダイレクト・メモリ・アクセスを使う必要はないことに注意．例えば，もしデータの送受信がI／Oポートを通して行なわれるならば，次のリード，ライト操作の間，CBIOSは選択したDMAオフセットとベースから始まる128バイトの領域をメモリ・バッファとして使用するであろう.

READ

ドライブが選択され，トラック，セクターがセットされ，DMAオフセットとセグメント・ベースもセットされていると，READサブルーチンはこれ等のパラメータに基づいて1セクター・リードを行う．ALレジスタに次のエラー・コードを返す．

0　正常終了

1　回復不能のエラー発生

現在，CP／M-86は，リターン・コードとして0か0でない値のみを返す．つまり，ALレジスタの値が0であると，CP／M-86はディスク操作が正しく終了したとみなす．しかし，エラーが発生した場合，CBIOSはエラーが回復可能かどうか調べる為に少くとも10回試行すべきである．

エラーの場合，BDOSは〝BDOS　ERRON　X：BAD　SECTOR″というメッセージを出力する．オペレータは，エラーを無視するならば，RETURNキーを押下げればよいし，中止したければコントロールCを入力すればよい．

WRITE

現在選択されているDMAバッファからのデータを，現在選択されているドライブ，トラック，セクターへ書く．データは他のCP／Mシステムとの互換性を保つ為に〝ノンデリーテッド・データ″として記録されるべきである．エラー・コードは，READコマンドと同様にALレジスタへ返され，エラー回復は上に述べたように行なう．

LISTST

リスト装置の準備完了のステイタスを返す．リスト装置が文字を受けつける用意ができていないと，ALレジスタへ00が，また文字をプリンタへ送ることができるときには0FFHを返す．

SECTRAN

CP／M-86の全応答を改善するために，論理セクターから物理セクターに変換する．標準CP／M-86システムは，〝スキュー・ファクター（skew facter″）6を持っている．5つの物理セクターが，連続するリード，ライト操作の間とびこされる．

このスキュー・ファクターは，ほとんどのプログラムが，次のセクターをぬかさないでそのバッファをロードするのに充分な時間をセクター間においている．速いプロセッサ，メモリ，ディスク・サブシステムを使用しているコンピュータ・システムでは，全応答を改善するために，スキュー・ファクターが変更されるかもしれない．しかしながら，コンピュータ・システム内部，または外部へ情報を移すために，単密度のIBMフォーマットのディスケットの場合には，スキューファクターは6に設定することが望ましい．

一般的に，SECTRANはCXに論理セクター番号を受けとる．この論理セクター番号は，0からセクター　－1の範囲の数である．またDXに変換表のアドレスを受けとる．セクター番号は，変換表の索引として使用され，BXに索引後の物理セクター番号を入れる．

標準システムでは，変換表と索引コードは，CBIOSに用意されており，変える必要はない．DX=0000Hであると，変換は行なわれずCXを単にBXにコピーして返す．他のときには，SECTRANは計算して，BXに変換したセクター番号を返す．ディスク・パラメータ・ヘッダーに変換表のアドレスが指定されていないときにも，SECTRANが呼ばれることに注意．

SETDMAB

CXレジスタは，連続するDMAリード，ライト操作の為のセグメント・ベースが入っている．BIOSはリード，ライト操作の間DMAベースとDMAオフセットによって決められたメモリ・アドレスから128バイト・バッファを使用する．

GETSEGB

BXにメモリ領域（Region）テーブル（MRT）の番地を返す．この値は，オペレーティング・システムの開始アドレスに対するテーブルのオフセットである．テーブルは，トランジェント・プログラムが利用できる物理メモリの位置と大きさを定義している．

インタラプト・ベクトルとCP／M-86オペレーティング・システム用に確保されたメモリ領域は，MRTに含まれていない．メモリ領域テーブルは，次の形式をしている．

	8 bit	
MRT：	R-Cnt	
0：	R-Base	R-Length
1：	R-Base	R-Length
	・・・	
n：	R-Base	R-Length
	16-bit	16-bit

R-Cnt は，メモリ領域記述子（上図のn＋1に等しい）の数である．一方R-BASEとR-Lengthは，各物理的な連続する領域の先頭と長さを示している．インタラプト用に確保された領域（通常0-3FFH）と，CP／M-86オペレーティング・システムは，このテーブルには含まれていないということを強調しておく．つまり，このテーブルにはトランジェント・プログラムで使用できる領域も含んでいるからである．

メモリが全て連続していると，R-Cnt フィールドは1，n＝0であり，領域を定義するメモリ領域記述子も1つだけとなる．

GETIOB　　ALに物理入出力装置バイト（IOBYTE）に対する論理値を返す．この8ビット値は，CP／M-86の4つの論理装置に関連した物理装置に使用される．

SETIOB　　BIOSにあるIOBYTEに値をセットするためにCLの値を使用する．

次章にBIOSの各種サブルーチンによって参照されるディスク・パラメータ・テーブルの形式と構成を述べる．

第6章
BIOSディスク定義テーブル

CP／M-80と同様に，CP／M-86は分離したフィルド構成である基本入出力システム（BIOS）を持つテーブル・ドリブンのオペレーティング・システムである．前の章で述べた様に，BIOS内の特定のサブルーチンを変更することによって，CP／M-86はRAMに基づいた8086，8088マイクロプロセッサ・システムでの操作を可能にする．

この章で述べることはCP／M-86 で使用する特別なディスク・システムの特性を定義するBIOS内のテーブルの概要と構成である．これ等のテーブル類は，自身でコーディングするか，CP／M-86で用意しているGENDEFユーティリティを使用して自動的に生成するかのどちらかである．これ等のテーブルの項目を以下に示す．

6.1 ディスク・パラメータ・テーブル

一般的に，各ディスク・ドライブは16バイトのディスク・パラメータ・ヘッダーを持っている．これには，ディスク・ドライブと，あるBDOS操作で必要となるスクラッチパド領域についての情報を含んでいる．

各ドライブに対するディスク・パラメータ・ヘッダーの形式を以下に示す．

Disk Parameter Header							
XLT	0000	0000	0000	DIRBUF	DPB	CSV	ALV
16b	16b	16b	16b	16b	16b	16b	16b

ただし，各項目は1ワード（16ビット）の値である．ディスク・パラメータ・ヘッダー（DPH）の各項目の意味を表6-1に示す．

表6-1　ディスク・パラメータ・ヘッダー項目

項　目	内　　容
XLT	必要ならこのドライブに対する論理番号から物理番号への変換ベクトルのオフセットである．また，もしセクター変換が必要でない時（例えば，物理セクターと論理セクターが同じ）には，0000Hの値が入る．同じセクター・スキューファクターを持つディスク・ドライブは，同じ変換表を共有する．
0000	BDOS内で使用するスクラッチパド値が入る（初期値は重要でない）．
DIRBUF	BDOS内のディレクトリ操作のための128バイトスクラッチパド領域のオフセット．全てのDPHのアドレスは同じスクラッチパド領域を指す．
DPB	このドライブに対するディスク・パラメータ・ブロックのオフセット．同じディスク特性を持つドライブは，同じディスク・パラメータ・ブロックを指す．
CSV	ディスクを変えるときにソフトウェアチェックで使われるスクラッチパド領域のオフセット．このオフセットは各DPHで異なる．
ALV	ディスク領域割当情報を保つためにBDOSによって使われるスクラッチパド領域のオフセット．このオフセットは各DPHにより異なる．

ディスク・ドライブがn個あるとき，DPHは次のような形をとる．すなわち，16バイトの最初の行がドライブ0に対応し，最後の行がn-1に対応しているテーブルである．そのテーブルは次のとおりである．

DPBASE

00	XLT 00	00 00	00 00	00 00	DIRBUF	DBP 00	CSV 00	ALV 00
01	XLT 01	00 00	00 00	00 00	DIRBUF	DBP 01	CSV 01	ALV 01
					:			
n-1	XLT n-1	00 00	00 00	00 00	DIRBUF	DBP n-1	CSV n-1	ALV n-1

ここでラベルDPBASEは，オペレーティング・システムの先頭に対するDPHテーブルのオフセットを定義している．

SELDSKサブルーチン（前章で述べた）は，選択されているドライブのオペレーティング・システムの開始からのDPHのオフセットを返してくる．次のコーディングに見られる様に，そのテーブルのオフセットが返される．その際，選択されたドライブが存在しない場合には，0000Hが返される．

```
NDISKS   EQU     4       ;NUMBER OF DISK DRIVES
.......
SELDSK:
         ;SELECT DISK N GIVEN BY CL
         MOV     BX,0000H     ;READY FOR ERR
         CPM     CL,NDISKS    ;N BEYOND MAX DISKS?
         JNB     RETURN       ;RETURN IF SO
                              ;0 <= N < NDISKS
         MOV     CH,0         ;DOUBLE (N)
         MOV     BX,CX        ;BX = N
         MOV     CL,4         ;READY FOR * 16
         SHL     BX,CL        ;N = N * 16
         MOV     CX,OFFSET DPBASE
         ADD     BX,CX        ;DPBASE + N * 16
RETURN:  RET                  ;BX - .DPH (N)
```

変換ベクトル（XLT00からXLTn-1）はBIOS内の他の場所にあり，かつそれは論理セクター番号0からセクター・カウント-1に1対1に対応している．各ドライブのディスク・パラメータ・ブロック（DPB）は更に複雑である．DPB(1つまたはそれ以上のDPHによって番地づけられている）は，次のような一般形をしている．

SPT	BSH	BLM	EXM	DSM	DRM	AL0	AL1	CKS	OFF
16b	8b	8b	8b	16b	16b	8b	8b	16b	16b

ここで，各々は1バイトまたは1ワードの値である．それは各フィールドの下に8b，16bとして示してある．各フィールドの意味を表6-2に定義する．

表6-2 ディスク・パラメータ・ブロック定義

フィールド	定　義
SPT	トラック当りの全セクター数.
BSH	データ・アロケーション・ブロック・シフト・ファクターであり，データ・ブロックのアロケーションサイズによって定まる.
BLM	データ・ブロック・アロケーション・サイズによって決まるブロック・マスクである.
EXM	エクステント・マスクであり，データ・ブロック・アロケーション・サイズとディスク・ブロック数によって決まる.
DSM	ディスク・ドライブの全容量を定める.
DRM	このドライブに格納できるディレクトリ・エントリーの総数を定める.
AL0, AL1	確保するディレクトリ・ブロックを決める.
CKS	ディレクトリのチェック・ベクトルの大きさである.
OFF	(論理) ディスクの先頭にある確保済トラックの数である.

これ等のテーブル値は，GENDEFによって自動的につくられるが，各フィールドの要素を見直す必要があるので，必要なときクロス・チェックを行うことができる．BSHとBLMの値は，データ・アロケーション・サイズBLSを(暗黙)に決定する．しかしBLSはディスク・パラメータ・ブロック内のエントリーではない．BLSの値とBSHおよびBLHの値の関係は，次に示す表6-3の通りである．表中の値は10進数である．

表6-3 BLSに対するBSHとBLM値

BLS	BSH	BLM
1,024	3	7
2,048	4	15
4,096	5	31
8,192	6	63
16,384	7	127

EXMの値はBLSとDSMの値が256より小さいか，または255より大きいかの二つによって決まる．次のテーブルのとおりである．

表6-4 EXMの最大値

BLS	DSM<256	DSM>255
1,024	0	N／A
2,048	1	0
4,096	3	1
8,192	7	3
16,384	15	7

ＤＳＭの値は個々のドライブの持つ最大データ・ブロック数である（ＢＬＳ単位).(ＤＳＭ＋１)をＢＬＳ倍した積は,ドライブによって保持された総バイト数であり，当然物理的なディスク容量の範囲内になければならない（オペレーティング・システムのトラックは計数していない).

ＤＲＭエントリーはディレクトリ・エントリーより１小さい数であり，16ビットの値まで許される．又，ＡＬ０とＡＬ１の値はＤＲＭによって決定され，以下に示すように一連の16ビットの数値とみなすことができる．

ＡＬ０								ＡＬ１							
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15

ここで00はＡＬ０のバイトの高位ビットに対応し，15はＡＬ１のバイトの低位ビットに対応している．各ビット位置はディレクトリ・エントリー数に対するデータ・ブロックを示しており，全部で16データ・ブロックをディレクトリ・エントリーの為に割当てることができる（割当は，00から始って右へ15の位置まで順番にビットが立てられる)．各ディレクトリ・エントリーは32バイトで構成されており，表6-5に示す．

表6-5　ＢＬＳとディレクトリ・エントリー数

ＢＬＳ	ディレクトリ・エントリー
1,024	ビット数の 32倍
2,048	ビット数の 64倍
4,096	ビット数の128倍
8,192	ビット数の256倍
16,384	ビット数の512倍

従って，もしＤＲＭ＝127（128ディレクトリ・エントリー）でＢＬＳ＝1024である場合，４ブロックが必要であり，ブロック当り32ディレクトリ・エントリーがあることになる．この場合ＡＬ０の高位４ビットがセットされ，ＡＬ０＝０Ｆ０Ｈ，ＡＬ１＝00Ｈの値となる．

ＣＫＳの値は次のようにして決まる．即ち，もしディスクが取りはずし可能な場合には，ＣＫＳ＝(ＤＲＭ＋１)／４となる．ここでＤＲＭは先程のディレクトリ・エントリー数である．ディスクが固定式の場合は，ＣＫＳ＝０にセットされる（この場合，ディレクトリ・レコードはチェックされない).

最後にＯＦＦは，物理的なディスクの先頭へ位置づけるために必要なスキップすべきトラック数を定める．この値は，ＳＥＴＴＲＫが呼ばれるたびに自動的に加算され，オペレーティング・システム用として確保されているトラックをスキップしたり，または大容量ディスクを小さい区分された部分に分けるためのメカニズムとして用いることができる．

ＤＰＢの議論を完全にするためには，いくつかのＤＰＢは，そのディスク・ドライブの特性が同一であれば，同じＤＰＢに番地づけることが可能であることを思い起こせばよい．更にＳＥＬＤＳＫが機能するたびに，ＢＤＯＳはＤＰＢの値をローカル領域へコピーするので，新しいディスクは単にＤＰＨ中のポインタを変えることにより番地づけられるので，ＤＰＢを動的に変えることができる．

最後にＣＳＶとＡＬＶについての説明を行う．二つの番地は，ＢＩＯＳに続く初期化されていないメモリ領域を指している．その領域は各ドライブ毎に存在しなければならず，各領域の大きさはＤＰＢ中の値によって定まる．

ＣＳＶによって番地づけられた領域の大きさは，ＣＫＳバイトであり，この特定ドライブのためのディレクトリ・チェック情報を保持するのに充分な量である．たとえば，ＣＫＳ＝（ＤＲＭ＋１）／４ならば，ディレクトリ・チェック用に（ＤＲＭ＋１）／４バイト確保しなければならない．ＣＫＳ＝０なら領域を確保する必要がない．

ＡＬＶで番地づけられた領域の大きさは，この特定のディスクに許されているデータ・ブロックの最大数によって決まり，それは次式で計算される．

（ＤＳＭ／８）＋１

付録Ｄに載せてあるＢＩＯＳは，標準８インチ単密度ドライブに対するこれらのテーブルの例を示している．このプログラムを調べ，上で定義したものとテーブル上の値とを比較することによってここで述べた事を確認してみるとよい．

6.2 GENDEFによるテーブル・ジェネレーション

CP/M-86で提供しているGENDEFユーティリティは，テーブル作成過程を大変容易にする．GENDEFはディスク定義命令を含むX．DEFファイルを読み，特定のドライブ構成をサポートするのに必要なテーブルを定義するアセンブラ言語命令を含むアウトプット・ファイル・X．LIBを作成する．GENDEFコマンドは次の形式である．

GENDEF　Xパラメータ・リスト

ここで，Xはファイル・タイプ（特に指定しなくてもよい）がDEFというファイルである．パラメータ・リストは表6-6で定義するシンボルをなにも含まないか，1つ以上により構成される．

表6-6　GENDEFオプション・パラメータ

パラメータ	効　　果
$C	ディスク・パラメータ・コメントを生成する
$O	DPBASE　OFFSET$を生成する
$Z	Z80，8080，8085用のソースを生成する
$COZ	（上記の全て）

Cパラメータは，各々定義したディスクの特性を記述する〝STAT　DSK:〟ユーティリティからの出力と同様なコメント行を，GENDEFに作らせる．通常DPBASEは，前に示したSELDSKサブルーチンの中で，MOV　CX，OFFSET DPBASEを必要とする．

```
DPBASE    EQU    $
```

として定義される．便利なように，$Oパラメータは，特別なプログラミング慣習との調整の為に，SELDSK中でMOV CX，DPBASEを許す定義，

```
DPBASE    EQU    OFFSET    $
```

を作成する．$Zパラメータは Z80，8080，8085アセンブラでアセンブルできるテーブルを生成する為に，標準8086/8088モードを無視する場合に用いられる．

X．DEFに含まれているディスク定義内容は，CP/Mテキストエディタで組込まれ，CP/M-80バージョン2のDISKDEFマクロで受けいれられるのと同一のディスク定義命令より成っている．BIOSディスク定義は，次のような一連のステートメントから成っている．

```
DISKS      n
DISKDEF    0, . . .
DISKDEF    1, . . .
. . . .
DISKDEF    n—1
. . . .
ENDEF
```

各ステートメントはキーワード，数字とデリミッタ間を任意のコメントをはめ込んだ一行から成っている．

ＤＩＳＫＳステートメントはシステムに構成されるドライブの数を定義している．ここで ｎは１～16の範囲の整数である．それから一連のＤＩＳＫＤＥＦマクロ呼出しが続く．それらは各論理ディスクの０からｎ－１（論理ドライブＡ～Ｐに対応している）の特性を定義している．ＤＩＳＫＳやＤＩＳＫＤＥＦステートメントは，前の節で述べたイン・ライン固定テーブルを生成する．そしてＢＩＯＳの非実行部分（通常ＢＩＯＳの最後，初期化されていないＲＡＭの先頭の前）におかねばならないことに注意しなければならない．

ＥＮＤＥＦ（End-of-Diskdet）ステートメントは，ＢＩＯＳの初期化されたＲＡＭを越えて配置される必要な未初期化ＲＡＭ領域を作成する．

ＤＩＳＫＤＥＦステートメントの形式は次の通りである．

ＤＩＳＫＤＥＦ　dn, fsc, lsc,〔Skf〕, bls, dks, dir, cks, ofs,〔０〕

ここで，

dn ：論理的ディスク番号で０～ｎ－１．
fsc：最初の物理セクター番号（０か１）
lsc：最後のセクター番号．
skf：オプションでセクター・スキュー・ファクタ．
bls：データ・アロケーション・ブロックの大きさ．
dks：bls単位でのディスクの大きさ．
dir：ディレクトリ・エントリーの数．
cks：チェックされるディレクトリ・エントリーの数．
ofs：論理トラック00に対するトラック・オフセット量．
〔0〕：オプションで1.4との互換性フラッグ．

〝dn〞の値は，ＤＩＳＫＤＥＦステートメントで定義しているドライブ数である．〝fsc〞パラメータにより，セクターの番号づけがわかる．通常fscは０または１である．〝lsc〞はトラック上の最後に番号付けされたセクターである．〝skf〞が指定されている場合，それはスキューに従って，セクター変換テーブルを作成するのに使われるセクター・スキュー・ファクターを定義する．もし，セクター数が256より小さいならば，１バイトテーブルが作られ，256以上の場合，各変換テーブル構成は２バイト必要である．skfパラメータが省略されているか０のときは，変換テーブルは作成されない．

〝bls〞パラメータは，各データ・ブロックへ割当てるバイト数を指定する．bls値としては，1024，2048，4096，8192，16384のいずれかがとられる．一般に，データ・ブロックは大きければ大きいほど，ディレクトリの参照が少くてすむので効率が上る．また，論理的に結びついたデータ・レコードは，物理的にもディスク上で近接している．さらに各ディレクトリ・エントリーが多くのデータを番地づければするだけ，ＢＩＯＳワーク領域が減少する．

〝dks〞は〝bls〞単位で全ディスク・サイズを指定する．すなわち，bls＝2048 でdks＝1000であれば，全ディスク容量は2,048,000バイトである．dksが255より大きいと，ブロック・サイズ・パラメータbls は1024より大きくなければならない．〝dir〞の値は，ディレクトリ・エントリーの総数を示し，必要であれば255を越えてもよい．

〝cks〞パラメータは，各ディレクトリを検索する場合に調べるディレクトリ・アイテム数を決定し，かつシステム稼動中にディスクが交換されたかどうかを検出するために内部的に用いられる（ディスクが交換された場合は，データが破壊されないようにCP／M-86は，次のコールド・スタート又はシステム・リセットが発生するまで自動的にリード・オンリー（書込み禁止）の印をつける）．

前の節で述べたように，フロッピー・ディスク・サブシステムの場合と同じように媒体（ディスク）が簡単に交換されるときは，cksの値はdirと等しくなる．ディスクが永久にマウントされてしまうときは，リスタートなしにディスクを交換する確率は非常に低いのでcksの値は０である．

〝ofs〞パラメータの値は，当該ドライブの値がアドレス付されたときスキップすべきトラック数を示しており，これはオペレーティング・システムのための追加領域を確保したり，一つの大容量の物理的なドライブ上に数個の論理ディスクを割当てるめに用いることができる．

最後に，高密度ディスク用にモディファイされたＣＰ／Ｍ 1.4とファイルの互換性が必要であるとき,〔０〕パラメータが指定される．このパラメータはディレクトリ・コンプレッションが起らないということを保証している．ディレクトリ・コンプレッションが起こると，このモディファイされたＣＰ／Ｍ 1.4とは互換性がなくなってしまう．通常はこのパラメータは指定されない．

便利さ，およびテーブル領域を節約するために，次の特別な形式

```
DISKDEF  i, j
```

があり，これによりディスクiは前に定義したドライブJと同じ特性になる．標準4ドライブ単密度システム（これはCP／M-80バージョン1.4と互換性がある）とCP／M-80バージョン2の上方向互換性は，次のステートメントを用いることによって定義される．

```
DISKS      4
DISKDEF    0, 1, 26, 6, 1024, 243,64, 64, 2
DISKDEF    1, 0
DISKDEF    2, 0
DISKDEF    3, 0
ENDEF
```

すなわち，

①全ディスクは1トラック当り26セクターの同じパラメータ値（1～26と番号づけられる）を持ち，
②各アクセスごとに6セクタースキップし
③1データ・ブロックは1024バイト
④243Kバイトのディスク容量のために243データ・ブロックを用意する．
⑤チェックされたディレクトリ・エントリーは64個
⑥オペレーティング・システムのために2トラック用意する．

DISKSステートメントは，n個のディスク・パラメータ・ヘッダー（DPH）を生成する．それはDISKSによって生成されたDPHテーブル・アドレスDPBASEより始まっている．上に述べたように，各ディスク・ヘッダー・ブロックは16バイトあり，定義済の各ドライブに1対1で対応している．例えば，4ドライブ標準システムでは，DISKSステートメントは次のテーブルを作成する．

```
DPBASE   EQU  $
DPE0     DW   XLTO, 0000H, 0000H, 0000H, DIRBUF, DPBO, CSV0, ALV0
DPE1     DW   XLTO, 0000H, 0000H, 0000H, DIRBUF, DPBO, CSV1, ALV1
DPE2     DW   XLTO, 0000H, 0000H, 0000H, DIRBUF, DPBO, CSV2, ALV2
DPE3     DW   XLTO, 0000H, 0000H, 0000H, DIRBUF, DPBO, CSV3, ALV3
```

参照の目的のために0～3の各ドライブに，テーブルの先頭番地を示すDPHラベルが含まれている．ディスク・パラメータ・ヘッダーの値についての詳細は，前の節で既に述べてある．

チェックおよび配置のベクトル番地は，BIOSコードとテーブルに続くRAM領域に含むべくENDEFステートメントによって生成される．

"skf"（スキュー・ファクター）パラメータが省略（または0）されたとき，変換テーブルは省略され，そのディスクに対するディスク・パラメータ・ヘッダーのXLTのところに0000Hという値が入ることに注意すること．

論理セクターから物理セクターに変換するためのマクロ・コールの結果，SECTRANはDX=0000Hという変換テーブルのアドレスを受け取り，BXレジスタのCXから元の論理セクターを単に返すだけである．変換テーブルはskfパラメータが現われたとき作成され，そのテーブル・アドレス（0でない）は対応するDPHに入る．例えば，次に示すテーブルはDISKDEFマクロ呼出しで，標準スキュー・ファクターskfが6であるとき作成されたものである．

```
XLTO  EQU   OFFSET  $
      DB    1, 7, 13, 19, 25, 5, 11, 17, 23, 3, 9, 15, 21
      DB    2, 8, 14, 20, 26, 6, 12, 18, 24, 4, 10, 16, 22
```

ENDEFステートメントに続いて，初期化されていないデータ領域の数が定義される．これらのデータ領域は，コールド・スタート時にロードされるBIOSの一部である必要はないが，BIOSとオペレーティング・システムのメモリの最後との間で使用可能でなければならない．初期化されていないRAM領域の大きさは，ENDEFステートメントによって生成されるEQUステートメントによって決まる．標準4ドライブ・システムに対してENDEFステートメントは，

```
1C72H=  BEGDAT  EQU  OFFSET  $
        (data areas)
1DB0=   ENDDAT  EQU  OFFSET  $
013C=   DATSIZ  EQU  OFFSET  $-BEGDAT
```

を作り出す．これは初期化されないRAM領域が，オフセット1C72から始まり1DB0H−1で終わり，013CHバイトを占めていることを示している．システムがロードされた後は，これらのアドレスは自由に使用できるようにしておかなければならない．

修正後，ドライブの特性を調べるためにSTATプログラムを用いることができる．というのはSTATが，そのドライブに関する情報を解読するためにディスク・パラメータ・ブロックを使用しているからである．GENCMDの$Cパラメータによって LIBファイルに格納されるコマンドは，STATの出力と整合しているであろう．STATコマンドは次の形，

```
STAT   d:DSK:
```

であり，ドライブd（d=A，・・・P）に対するディスク・パラメータ・ブロックを解釈し，次に示す値を表示する．

```
r: 128 Byte Record Capacity
k: Kilobyte Drive  Capacity
d: 32  Byte Directory Entries
c: Checked  Directory Entries
e: Records/ Extent
b: Records/ Block
s: Sectors/ Track
t: Reserved Tracks
```

6.3 GENDEFの出力

GENDEFはコンソールでDEFファイルに入れるステートメントのリスティングを作る（必要なら，CONTROL- Pによりリスティングを印字できる）．各ソース行は番号付けされており，エラーがあればエラーを起こした条件を示すアイテムと〝?〟を行の下に示す．GENDEFが作り出すソース・エラーを，表6-7に記す．続いてインプット，アウトプット・ファイル作成する時に発生するエラーを表6-8に挙げる．

表6-7 GENDEFソース・エラー・メッセージ

メッセージ	意　味
Bad Val	DISKSステートメントで16以上のディスクが定義されている．
Convert	数をコンバートできない．ASM86で認めている2進，8進，10進，16進数でない．
Delimit	パラメータ間のデリミッタが誤っている．
Duplic	ディスク・ドライブに対して二重の定義をしている．
Extra	行の終りにエクストラ・パラメータが有る．
Length	キーワードやデータ・アイテムが長すぎる．
Missing	この位置にパラメータが必要である．
No Disk	参照したディスクがその前に定義されていない．
No Stmt	ステートメント・キーワードが正しくない．
Numeric	この位置で数値を必要としている．
Range	この位置での数値が範囲を越えている．
Too Few	パラメータが不足している．
Quote	現在行にエンドを引用した誤り．

表6-8 GENDEF インプット，アウトプット・エラー・メッセージ

メッセージ	意　味
Cannot Close ".LIB" File	LIBファイルのクローズ処理が不成功．通常ハードウェア書込み禁止により発生する．
"LIB" DISK FULL	LIBファイル用のスペースが足りない．
No Input File Present	指定されたDEFファイルが見つからなかった．
No ".LIB" Directory Space	LIBディスク上にあまりにも多数のファイルが有り，LIBファイルを作ることができない．
Premature End-of-File	DEFファイルの終りが，予期しないところにある．

次のステートメントを持っているファイルTWO．DEFがあるとき，

```
disks 2
diskdef  0 , 1 , 26, 6 , 2048, 256, 128, 128, 2
diskdef  1 , 1 , 58, 2048, 1024, 300, 0 , 2
endef
```

コマンド　gencnd two $C

は，コンソールへ次の出力をおこなう．

```
DISKDEF Table Generator, Vers 1.0
1              DISKS 2
2              DISKDEF  0 , 1 , 58, 2048, 256, 128, 128, 2
3              DISKDEF  1 , 1 , 58, 2048, 1024, 300, 0 , 2
4              ENDEF
No Error (S)
```

結果として，TWO．LIBファイルはASM-86 INCLUDE指令を使用して次に示すアセンブラ言語プログラムが格納される．このASM-86出力リストは，右側がつめられているが，GENDEFとASM-86を用いて簡単に再成できる．

```
                        ;       Sample Program Including TWO.LI
                        ;
                        SELDSK:
                        ;       ....
  0000 B9 03 00                 MOV     CX,OFFSET DPBASE
                        ;       ....
=                               INCLUDE TWO.LIB
=                       ;               DISKS   2
=   0003                dpbase  equ     $                   ;Base o
= 0003 32 00 00 00      dpe0    dw      xlt0,0000h          ;Transl
= 0007 00 00 00 00              dw      0000h,0000h         ;Scratc
= 000B 5B 00 23 00              dw      dirbuf,dpb0         ;Dir Bu
= 000F FB 00 DB 00              dw      csv0,alv0           ;Check,
= 0013 00 00 00 00      dpe1    dw      xlt1,0000h          ;Transl
= 0017 00 00 00 00              dw      0000h,0000h         ;Scratc
= 001B 5B 00 4C 00              dw      dirbuf,dpb1         ;Dir Bu
= 001F 9B 01 1B 01              dw      csv1,alv1           ;Check,
=                       ;               DISKDEF 0,1,26,6,2048,2
=                       ;
=                       ;       Disk 0 is CP/M 1.4 Single Densi
=                       ;        4096:  128 Byte Record Capacit
=                       ;         512:  Kilobyte Drive  Capacit
=                       ;         128:  32 Byte Directory Entri
=                       ;         128:  Checked Directory Entri
=                       ;         256:  Records / Extent
=                       ;          16:  Records / Block
=                       ;          26:  Sectors / Track
=                       ;           2:  Reserved  Tracks
=                       ;           6:  Sector Skew Factor
=                       ;
=   0023                dpb0    equ     offset $            ;Disk P
= 0023 1A 00                    dw      26                  ;Sector
= 0025 04                       db      4                   ;Block
= 0026 0F                       db      15                  ;Block
```

```
= 0027 01                           db      1               ;Extnt
= 0028 FF 00                        dw      255             ;Disk S
= 002A 7F 00                        dw      127             ;Direct
= 002C C0                           db      192             ;Alloc0
= 002D 00                           db      0               ;Allocl
= 002E 20 00                        dw      32              ;Check
= 0030 02 00                        dw      2               ;Offset
=   0032                   xlt0     equ     offset $        ;Transl
= 0032 01 07 0D 13                  db      1,7,13,19
= 0036 19 05 0B 11                  db      25,5,11,17
= 003A 17 03 09 0F                  db      23,3,9,15
= 003E 15 02 08 0E                  db      21,2,8,14
= 0042 14 1A 06 0C                  db      20,26,6,12
= 0046 12 18 04 0A                  db      18,24,4,10
= 004A 10 16                        db      16,22
=   0020                   als0     equ     32              ;Alloca
=   0020                   css0     equ     32              ;Check
=                                           DISKDEF 1,1,58,,2048,10
=                          ;
=                          ;
=                          ;        Disk 1 is CP/M 1.4 Single Densi
=                          ;        16384:  128 Byte Record Capacit
=                          ;         2048:  Kilobyte Drive  Capacit
=                          ;          300:  32 Byte Directory Entri
=                          ;            0:  Checked Directory Entri
=                          ;          128:  Records / Extent
=                          ;           16:  Records / Block
=                          ;           58:  Sectors / Track
=                          ;            2:  Reserved  Tracks
=                          ;
=   004C                   dpbl     equ     offset $        ;Disk P
= 004C 3A 00                        dw      58              ;Sector
= 004E 04                           db      4               ;Block
= 004F 0F                           db      15              ;Block
= 0050 00                           db      0               ;Extnt
= 0051 FF 03                        dw      1023            ;Disk S
= 0053 2B 01                        dw      299             ;Direct
= 0055 F8                           db      248             ;Alloc0
= 0056 00                           db      0               ;Allocl
= 0057 00 00                        db      0               ;Check
= 0059 02 00                        dw      0               ;Offset
=   0000                            dw      2               ;No Tra
=   0080                   xltl     equ     0               ;Alloca
=   0000                   alsl     equ     128             ;Check
=                          cssl     equ     0
=                                           ENDEF
=                          ;
=                          ;
=                          ;        Uninitialized Scratch Memory Fo
=   005B                   ;
= 005B                     begdat   equ     offset $        ;Start
= 00DB                     dirbuf   rs      128             ;Direct
= 00FB                     alv0     rs      als0            ;Alloc
= 011B                     csv0     rs      css0            ;Check
= 019B                     alvl     rs      alsl            ;Alloc
=   019B                   csvl     rs      cssl            ;Check
=   0140                   enddat   equ     offset $        ;End of
= 019B 00                  datsiz   equ     offset $-begdat ;Size o
                                    db      0               ;Marks
                                    END
```

第7章
CP／M-86ブートストラップと改造手続き

この章では，標準CP／M-86提供ディスクの構成，各構成の動きと非標準ハードウェアにCP／M-86を採用するときの手続きを述べる．

CP／M-86は，先のCP／M-80オペレーティング・システムと互換性のあるファイル形式を使用している単密度のIBMと互換性のある8インチ・ディスケットで供給される．特に，最初の2トラックは，オペレーティング・システムとブートストラップ・プログラムのために確保されている．同時にディスケットの残りには，プログラム，データ・ファイルを指しているディレクトリ情報がある．CP／M-86は，インテル204コントローラを通してフロッピー・ディスクに接続されたインテルSBC 86／12 シングル・ボード・コンピュータで動作するように供給されている．この構成のCP／M-86のオペレーションは，他の8086や8088環境でのモデルに対しても同様である．

供給システムの主構成を，以下に示す．

・86／12ブートストラップ・ROM（BOOTROM）
・コールド・スタート・ローダー（LOADER）
・CP／M-86システム（CPM．SYS）

SBC86／12にインストールされると，BOOTROMはロケーション・バイト0FF000Hから始まるメモリ・アドレス・スペースの一部になる．非標準環境では，同等のイニシャル・ローダがBOOTROMの代わりに必要である．従って，ROM自身はCP／M-86と一緒には含まれない．Digital ResearchからBOOTROMを得ることができる．または，附録Cに挙げたリスティングかBOOT．A86として供給ディスクにあるソース・ファイルからプログラム化することができる．BOOTROMの行うことは，最初の2トラック（システム用）からメモリへLOADERを読込んで，実行のためにLOADERへプログラム制御を渡す．

7.1 コールド・スタート・ロード・オペレーション

LOADERプログラムは，システム・ディスクからメモリへCPM．SYSを読込むファイル処理機能のあるCP／M-86の単純なバージョンである．LOADERが処理を終了すると，CPM．SYSプログラムは制御を受けオペレータの入力コマンド処理をはじめる．

LOADERとCPM．SYSプログラムとも，標準CMDヘッダ・レコードが先行している．128バイトのLOADERヘッダー・レコードは，次のようなシングル・グループ記述子がある．

G-Form	G-Length	A-Base	G-Min	G-Max
1	xxxxxxxxx	0400	xxxxxxx	xxxxxxx
8b	16b	16b	16b	16b

ここで，G-Form＝1はコード・グループを示している．"X"フィールドは無視であり，A-BaseはBOOTROMがメモリをに対して書込みを 始めるパラグラフ・アドレスを定義している（A-BASEはヘッダの開始から3バイトをオフセットするワード値である）．コード・グループだけがあるときは，8080メモリ構成であると仮定することに注意．さらに，A-BASEはLOADER（バイト・アドレス0400H）に対するベース・パラグラフ・アドレスを定義し，事実LOADERはCP／M-86かBOOTROMをオーバラップしないパラグラフ境界にロードされ実行できる．

LOADER自身は，三つの部分より構成されている．即ち，Load CPM Program（LDCPM），Loader Basic Disk System（LDBDOS）とLoader Basic I／O System（LDBIOS）である．LOADERはインテル86／12構成を使用するCP／M-86をイニシャライズするためにセットアップするけれども，LDBIOSは前章のBIOSの改造で述べたのと同じエントリー・ポイントを使用する非標準ハードウェアに対してフィールド変換をすることができる．LOADERの構成を図7-1に示す．

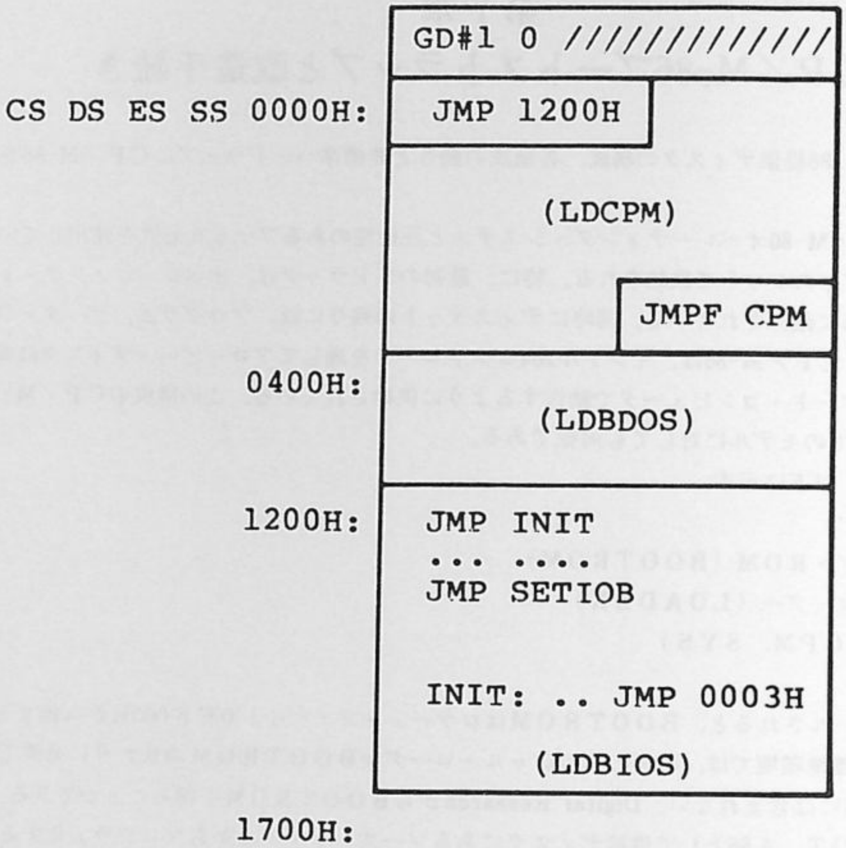


図7-1　ＬＯＡＤＥＲ構成

ベース・レジスタからのバイト・オフセットは，図の左側に示してある．ＧＤ＃１は先に述べたようにＬＯＡＤＥＲコード・グループに対するグループ・記述子であり，すぐ〝0″グループ・ターミネータが続いている．ＢＯＯＴＲＯＭはヘッダ・レコードを除いてA-フィールドで与えられる04000Ｈから全ＬＯＡＤＥＲプログラムを読込む．読込みの終了後，ＢＯＯＴＲＯＭはＬＯＡＤＥＲプログラムが実行を開始するロケーション04000Ｈへ制御を渡す．ＬＤＣＰＭのベースにあるＪＭＰ　1200Ｈ命令は，ＬＤＢＩＯＳの先頭にあるＩＮＩＴサブルーチンへ制御を渡す．ＩＮＩＴで開始するサブルーチンは装置の初期化，サイン・オン・メッセージを印刷し，バイト・オフセット0003ＨのＬＤＣＰＭプログラムへ制御を戻す．ＬＤＣＰＭモジュールはＣＰＭ．ＳＹＳファイルをオープンし，メモリにＣＰ／M-86システムをロードして，ＬＤＣＰＭ実行の最後にあるJMPF　ＣＰＭによってＣＰ／M-86へ制御をうつす．この様にしてコールド・スタート・シーケンスは完了する．

ＬＤＢＩＯＳを改造し，それをＬＯＡＤＥＲに組み込むことができるようにファイルＬＤＣＰＭ．Ｈ86とＬＤＢＤＯＳ．Ｈ86がＣＰ／M-86に含まれている．事実，ＢＩＯＳ．Ａ86には〝loader bios″と呼ぶ条件アセンブリ・スイッチがあり，ＬＤＢＩＯＳを作ることが可能である．ＬＤＢＩＯＳのＩＮＩＴサブルーチンの部分を，参考として附録Ｃに載せておく．カスタムＬＤＢＩＯＳを構築するためには，通常のＢＩＯＳをオフセット1200Ｈから始まるようにし，ディスクと装置の初期化を行う為にＩＮＩＴで開始する初期化サブルーチンをＩＮＩＴサブルーチンの最後にあるオフセット0003Ｈへのジャンプを含めて，変える必要がある．ＬＤＢＩＯＳ．Ａ86プログラムをアセンブルするにはＡＳＭ-86を使用する．即ち

ＡＳＭ86　ＬＤＢＩＯＳ

は，ＬＤＢＩＯＳ．Ｈ86マシン・コード・ファイルを作成する．ＰＩＰを使用して3つのＬＯＡＤＥＲモジュールをつなぎ合せる．即ち，

ＰＩＰ　ＬＯＡＤＥＲ．Ｈ86＝ＬＤＣＰＭ．Ｈ86，ＬＤＢＤＯＳ．Ｈ86，ＬＤＢＩＯＳ．Ｈ86

これにより，ＬＯＡＤＥＲプログラムのマシン・コード・ファイルを作る．標準ＬＯＡＤＥＲプログラムはオフセット1700Ｈで終っているけれども，改造したＬＤＢＩＯＳは，ＬＯＡＤＥＲが最初の２トラック内にあり，ＣＰ／Ｍ-86領域にオーバラップしていないという制限でこの最終番地は異なるかもしれない．ＧＥＮＣＭＤユーティリティを使用してＬＯＡＤＥＲのコマンド・ファイルを作る．即ち，

ＧＥＮＣＭＤ　ＬＯＡＤＥＲ　8080　ＣＯＤＥ〔Ａ400〕

は400Ｈの絶対パラグラフ番地即ちバイト・アドレス4000Ｈを持つ8080メモリ・モデルを定義するヘッダー・レコードを有するファイルＬＯＡＤＥＲ．ＣＭＤを作る．8080システムのロケーション900ＨへＬＯＡＤＥＲ．ＣＭＤを読込む為にDDTを使用し，ディスクの最初の２トラックへローダをコピーする為に8080コーティリティＳＹＳＧＥＮを使用しなさい．

```
A>DDT
-ILOADER.CMD
-R800
-^C
A>SYSGEN
SOURCE DRIVE NAME (or return to skip) <cr>
DESTINATION DRIVE NAME (or return to skip) B
```

代りに，操作的にＣＰ／Ｍ-86・システムへアクセスするなら，コマンド，

ＬＤＣＯＰＹ　ＬＯＡＤＥＲ

がシステム・トラックへＬＯＡＤＥＲをコピーする．これでメモリへＣＰＭ．ＳＹＳファイルを読込むことのできるカスタムＬＤＢＩＯＳと一体になったＬＯＡＤＥＲプログラムのディスケットを持つことになる．標準としては，ＬＯＡＤＥＲはロケーション4000Ｈから実行すると考えられる．ＬＯＡＤＥＲはにリロケータブルである．しかしＬＯＡＤＥＲのオペレーティング・アドレスは，ヘッダー・レコードにあるA-Baseの値によってのみ決定される．

もちろん，メモリへＬＯＡＤＥＲを取り込むためにはＢＯＯＴＲＯＭと同じ機能を遂行しなければならない．ブート・オペレーションは，通常２通りのうちの１つが採用されている．第一に，コンピュータのリセット・ボタンが押されたとき，ＢＯＯＴＲＯＭと同じ機能を遂行するＲＯＭ（あるいはＰＲＯＭ）をプログラムできる．
次に大部分のコントローラは，メモリへ最初のディスク・セクターを読むパワー・オン・〝ブート〟・オペレーションを提供する．この１セクター・プログラムは，順に残りのセクターからＬＯＡＤＥＲ読み，終了するとＬＯＡＤＥＲへ制御を渡すことにより，ＢＯＯＴＲＯＭと同じ動きを遂行する．これ等のどちらもハードウェアの特性に依存するので，オペレーティング環境に照しあわせる必要がある．

7.2 CPM．SYSの構成

LOADERプログラムが読むCPM．SYSファイルは，LOADERプログラムと同じ様に128バイトのヘッダー・レコードがあるCMDファイル形式のCCP，BDOS，BIOSより成っている．即ち，

G-Form	G-Length	A-Base	G-Min	G-Max
1	xxxxxxxxx	040	xxxxxxx	xxxxxxx
8b	16b	16b	16b	16b

ここで，代っているのは，A-Baseロード・アドレスがパラグラフ040H（バイト・アドレス0400H）で，8086インタラプト・ロケーションに続いていることである．図7-2にディスク上でのCPM．SYSファイルの全てを示す．

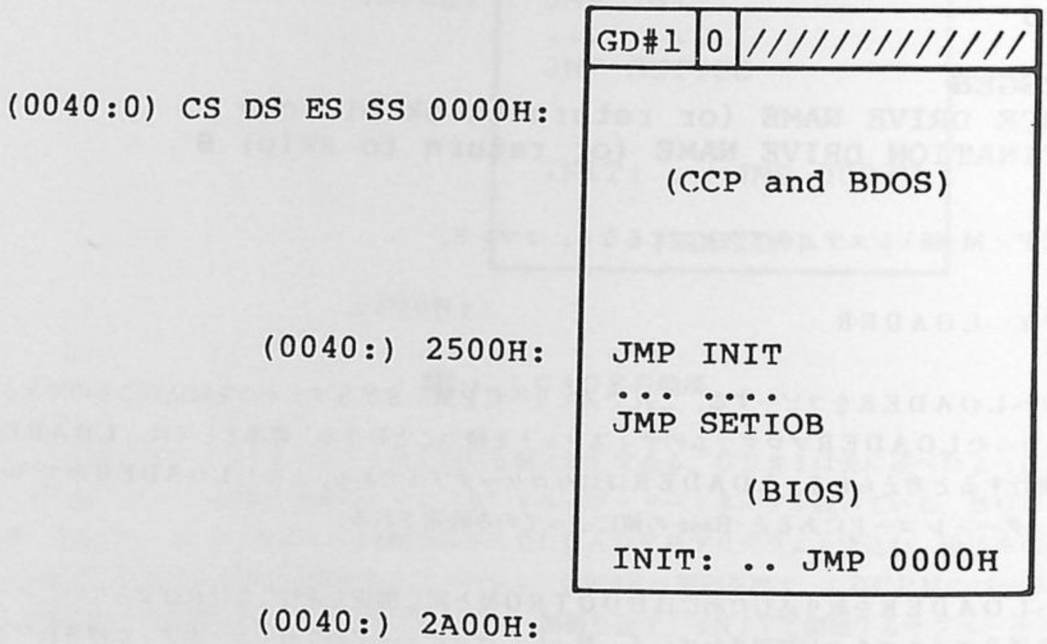


図7-2　CPM．SYSファイル構成

ここでGD＃1は，〝0〞ターミネータが続くA-Base値を含むグループ記述子である．86／12　BIOSは附録Dに載せてあるが，それにはディスク定義テーブルを含むSINGLES．LIBファイルを読む〝インクルード〞命令を持っている．SINGLES．LIBファイルは，以下に示すSINGLES．DEFを使用してGENDEFによって作られる．

```
disks   2
diskdef  0, 1, 26, 6, 1024, 243, 64, 64, 2
diskdef  1, 0
endef
```

CPM．SYSファイルは，LOADERプログラムによってA-Base（バイト・アドレス0400H）で示される番地へ読込まれ，制御はオフセット・アドレス2500HにあるINITエントリー・ポイントへ渡される．LOADERが行なわない幾つかの初期化をINITサブルーチンで行い，完了後INITはCCPの実行を開始する為にJMP0000Hを実行する．CPM．SYSの実際のロード・アドレスは，メモリの別の領域でCP／M-86を実行したいなら変えることができる．A-Base フィールドで示される番地により決定される．

オペレーティング・システムがしめる領域は，BIOSメモリ領域テーブルから除かねばならないことに注意せよ．

ＬＯＡＤＥＲプログラムと同様にソース・ディスクに含まれるＢＩＯＳ．A86か基本となるＣＢＩＯＳ．A86アセンブリ言語ファイルのどちらかを変えることによってＢＩＯＳを改造することができる．いずれにおいても，特別なＩ／Ｏドライバーに合せたＢＩＯＳを作成し，ＡＳＭ-86を使用してアセンブルする．即ち，

ＡＳＭ86　ＢＩＯＳ

は，ＢＩＯＳマシン・コードを含むＢＩＯＳ．H86ファイルを作る．

供給ディスク上のＣＰＭ．H86ファイルへこの新しいＢＩＯＳを連結するには，

ＰＩＰ　ＣＰＭＸ．H86＝ＣＰＭ．H86，ＢＩＯＳ．H86

を実行する．このＣＰＭＸ16進ファイルを実行形式のＣＭＤファイルへ変換するには，

ＧＥＮＣＭＤ　ＣＰＭＸ　8080　ＣＯＤＥ〔A40〕

により，Ａ-Base＝40Ｈをもつ ＣＭＤメモリ・イメージを作成し，次のＲＥＮコマンドによりＣＰＭＸファイルをＣＰＭへリネームし，8086システム・ディスクへ格納する．

ＲＥＮ　ＣＰＭ．ＳＹＳ＝ＣＰＭＸ．ＣＭＤ

これで，カストマイズが完了したことになる．そして，カストマイズＢＯＯＴＲＯＭか，バイト・ロケーション04000Ｈへ ＬＯＡＤＥＲプログラムを読込む１セクター・コールド・スタート・ローダのいずれかに ＢＯＯＴＲＯＭをおきかえる．ＬＯＡＤＥＲプログラムは順に，カスタムＢＩＯＳをもつＣＰＭ．ＳＹＳファイルをバイト・ロケーション0400Ｈのメモリへ読込む．ＣＰ／Ｍへ制御を移し，オペレーションを開始する．ＣＰ／Ｍ-86は，次のコールド・スタート・オペレーションが行なわれるまでメモリ上にとどまる．

システム・トラック上に完全なＣＰＭ．ＳＹＳファイルを保持するに十分な領域をもつ非標準ディスクを構成するなら，この２段階のブート・オペレーションをさけることができる．この場合，コールド・スタートは，メモリのA-Baseによって示されるロケーションへＣＰ／Ｍ-86メモリ・イメージをロードし，オフセット2500ＨのＩＮＩＴエントリー・ポイントへ制御を渡す．このように，中間のＬＯＡＤＥＲプログラムは，ＬＤＢＩＯＳによる初期化を完全に省略する．もちろん，ＢＩＯＳがそれにとってかわる．

ＡＳＭ-86，ＧＥＮＣＭＤとＧＥＮＤＥＦは，ＣＰ／Ｍ-80またはＣＰ／Ｍ-86のいずれのカストマイズ処理にも使用しうるよう，ＣＯＭ，ＣＭＤ形式の相方が提供されている．たとえCP／M-80，CP／M-86のいずれも走っていない環境でも最低限のエディタとデバック・ツールをもっているならば，ＣＰＭ．ＳＹＳファイルと同様のシステム・トラックを読み書きする特殊なディスクＩ／Ｏルーチンを書くことができる．

２つのシステム・トラックは単にアクセスするだけであるが，ＣＰＭ．ＳＹＳファイルをリードするには多少の難しさがある．ＣＰＭ．ＳＹＳはディスク上の最初のファイルであり，ディスケットのディレクトリにしたがってただちに現われる．ディレクトリは，ディスケットの第３トラックより始まり，最初の16論理セクターの占有する．セクターは，ディレクトリ・トラックよりファクター６でスキューされる．それ故に，ＣＰＭ．ＳＹＳファイルの読込みは６セクター毎にロードされねばならない．明らかに，既存のＣＰ／Ｍシステムを使用してこのコンバージョンを行う方が，時間的にも効果的である．

付録A
セクターのブロッキングおよびデブロッキング

BIOSのWRITEエントリー・ポイントの呼出しに対して，CP/M-86のBDOSはホスト・ディスク・サブシステムが128バイトの倍数のセクター・サイズを持っているときに，効果的なセクター・ブロッキングおよびデブロッキングを可能にする情報を引き渡す．

この付録の目的は，BDOSから引き渡される情報を用いてオペレーションを自動的に行う汎用的なアルゴリズムを提供することである．

WRITEの各呼出しに関して，BDOSはレジスタCLに，次の情報を用意する．

0＝通常なセクター書込み
1＝ディレクトリ・セクターの書込み
2＝新しいデータ・ブロックの最初のセクターへの書込み．

状態0は次のような場合に起きる．すなわち次の書込み動作が（たとえば，ランダム・モードでレコードを更新するときのように）以前に書込まれていた領域になされるとき，書込みが未割当てブロックの最初のセクター以外であるとき，またはディレクトリ領域以外になされるときである．状態1は書込みがディレクトリ領域に対してなされるときに起きる．状態2は新しく割振られたデータ・ブロックの，最初のレコードに書かれるとき発生する．ほとんどの場合，適用プログラムは128バイトの倍数のセクター単位で順次読み書きを行う．この結果，レコードを書込む場合先読みが必要でなくなるので，レコードをブロッキングまたはデブロッキングするとき，読み書きのどちらのオペレーションにおいてもオーバヘッドはほとんど生じない．

この付録は，ブロッキング，デブロッキングのアルゴリズムの骨格を示している（このファイルはCP/M-86ディスクに含まれている）．一般にそのアルゴリズムは，CP/Mのセクター読込みオペレーションの全てを，ホスト・ディスクのサイズと等しい中間バッファを通して，ホスト・ディスク上に割振るというものである．そのプログラム中のシーク・オペレーションに含まれるCP/Mセクターに関連する全ての値（定数）および変数には〝sek〟という接頭語がついており，一方ホスト・ディスク・システムに関係する定数および変数には〝hst〟という接頭語がついている．付録Fの24行から始まるEQUステートメントは，CP/Mとホスト・システムとの割振りを定義しており，必要に応じて変えなければならない．

SELDSKエントリー・ポイントは，新しいディスクがログインされた時，ホスト・バッファ・フラッグをクリアする．SELDSKエントリー・ポイントはディスク・パラメータ・ヘッダーのアドレスを計算して返してくれるが，この時点では物理的にホスト・ディスクを選択してはいない（それは後でREADHSTかWRITEHSTによって選択される）．更にSETTRK，SETSEC，SETDMAは単に値を保存するだけで，この時点では他のどんな動作もしない．SECTRANは単に物理的なセクター番号を返すだけである．

重要なエントリー・ポイントはREADとWRITEである．これらのサブルーチンがユーザーのREADまたはWRITEオペレーションと置き換わる．

実際の物理的な読込み，または書込みはREADHSTまたはWRITEHSTのところで行なわれる．そこではすべての値が用意されている．hst dskはホスト・ディスク番号，hst trkはホスト・トラック数，hst secはホスト・セクター番号(これは物理的なセクター番号への変換が必要となるであろう)．

この時点でhst sizの大きさを持つhst bufというバッファからホスト・セクターを読込んだり書込んだりする完全なコードを挿入しなければならない．他の全ての割付け機能は，そのアルゴリズムによっておこなわれる．

```
;*****************************************************
;*                                                   *
;*          Sector Blocking / Deblocking             *
;*                                                   *
;* This algorithm is a direct translation of the     *
;* CP/M-80 Version, and is included here for refer-  *
;* ence purposes only.  The file DEBLOCK.LIB is in-  *
;* cluded on your CP/M-86 disk, and should be used   *
;* for actual applications.  You may wish to contact *
;* Digital Research for notices of updates.          *
;*                                                   *
;*****************************************************
;
;*****************************************************
;*                                                   *
;*         CP/M to host disk constants               *
;*                                                   *
;* (This example is setup for CP/M block size of 16K *
;* with a host sector size of 512 bytes, and 12 sec- *
;* tors per track.  Blksiz, hstsiz, hstspt, hstblk   *
;* and secshf may change for different hardware.)    *
;*****************************************************
una     equ     byte ptr [BX]       ;name for byte at BX
;
blksiz  equu    16384               ;CP/M allocation size
hstsiz  equ     512                 ;host disk sector size
hstspt  equ     12                  ;host disk sectors/trk
hstblk  equ     hstsiz/128          ;CP/M sects/host buff
;
;*****************************************************
;*                                                   *
;* secshf is log2(hstblk), and is listed below for   *
;* values of hstsiz up to 2048.                      *
;*                                                   *
;*              hstsiz    hstblk    secshf           *
;*                256        2         1             *
;*                512        4         2             *
;*               1024        8         3             *
;*               2048       16         4             *
;*                                                   *
;*****************************************************
secshf  equ     2                   ;log2(hstblk)
cpmspt  equ     hstblk * hstspt     ;CP/M sectors/track
secmsk  equ     hstblk-1            ;sector mask
;
;*****************************************************
;*                                                   *
;*         BDOS constants on entry to write          *
;*                                                   *
;*****************************************************
wrall   equ     0                   ;write to allocated
wrdir   equ     1                   ;write to directory
wrual   equ     2                   ;write to unallocated
;
;*****************************************************
;*                                                   *
```

```
;*          The BIOS entry points given below show the   *
;*          code which is relevant to deblocking only.   *
;*                                                       *
;*********************************************************
seldsk:
            ;select disk
            ;is this the first activation of the drive?
            test DL,1                   ;lsb = 0?
            jnz selset
            ;this is the first activation, clear host buff
            mov hstact,0
            mov unacnt,0
selset:
            mov al,cl ! cbw             ;put in AX
            mov sekdsk,al               ;seek disk number
            mov cl,4 ! shl al,cl        ;times 16
            add ax,offset dpbase
            mov bx,ax
            ret
;
home:
            ;home the selected disk
            mov al,hstwrt               ;check for pending write
            test al,al
            jnz homed
            mov hstact,0                ;clear host active flag
homed:
            mov cx,0                    ;now, set track zero
;           (continue HOME routine)
            ret
;
settrk:
            ;set track given by registers CX
            mov sektrk,CX               ;track to seek
            ret
;
setsec:
            ;set sector given by register cl
            mov seksec,cl               ;sector to seek
            ret
;
setdma:
            ;set dma address given by CX
            mov dma_off,CX
            ret
;
setdmab:
            ;set segment address given by CX
            mov dma_seg,CX
            ret
;
sectran:
            ;translate sector number CX with table at [DX]
            test DX,DX          ;test for hard skewed
            jz notran           ;(blocked must be hard skewed)
            mov BX,CX
            add BX,DX
```

```
        mov BL,[BX]
        ret
no_tran:
        ;hard skewed disk, physical = logical sector
        mov BX,CX
        ret
;
read:
        ;read the selected CP/M sector
        mov unacnt,0                ;clear unallocated counter
        mov readop,1                ;read operation
        mov rsflag,1                ;must read data
        mov wrtype,wrual            ;treat as unalloc
        jmp rwoper                  ;to perform the read
;
write:
        ;write the selected CP/M sector
        mov readop,0                ;write operation
        mov wrtype,cl
        cmp cl,wrual                ;write unallocated?
        jnz chkuna                  ;check for unalloc
;
;       write to unallocated, set parameters
;
        mov unacnt,(blksiz/128)     ;next unalloc recs
        mov al,sekdsk               ;disk to seek
        mov unadsk,al               ;unadsk = sekdsk
        mov ax,sektrk
        mov unatrk,ax               ;unatrk = sektrk
        mov al,seksec
        mov unasec,al               ;unasec = seksec
;
chkuna:
        ;check for write to unallocated sector
;
        mov bx,offset unacnt        ;point "UNA" at UNACNT
        mov al,una ! test al,al     ;any unalloc remain?
        jz alloc                    ;skip if not
;
;       more unallocated records remain
        dec al
        mov una,al                  ;unacnt = unacnt-1
        mov al,sekdsk
        mov BX,offset unadsk        ;same disk?
        cmp al,una
        jnz alloc                   ;sekdsk = unadsk?
                                    ;skip if not
;
;       disks are the same
        mov AX, unatrk
        cmp AX, sektrk
        jnz alloc
                                    ;skip if not
;
;       tracks are the same
        mov al,seksec
                                    ;same sector?
;
        mov BX,offset unasec        ;point una at unasec
;
```

```
        cmp al,una                      ;seksec = unasec?
        jnz alloc                       ;skip if not
;
;       match, move to next sector for future ref
        inc una                         ;unasec = unasec+1
        mov al,una                      ;end of track?
        cmp al,cpmspt                   ;count CP/M sectors
        jb noovf                        ;skip if below
;
;       overflow to next track
        mov una,0                       ;unasec = 0
        inc unatrk                      ;unatrk=unatrk+1
;
noovf:
        ;match found, mark as unnecessary read
        mov rsflag,0                    ;rsflag = 0
        jmps rwoper                     ;to perform the write
;
alloc:
        ;not an unallocated record, requires pre-read
        mov unacnt,0                    ;unacnt = 0
        mov rsflag,1                    ;rsflag = 1
                                        ;drop through to rwoper
;
;*****************************************************
;*                                                   *
;*      Common code for READ and WRITE follows       *
;*                                                   *
;*****************************************************
rwoper:
        ;enter here to perform the read/write
        mov erflag,0                    ;no errors (yet)
        mov al, seksec                  ;compute host sector
        mov cl, secshf
        shr al,cl
        mov sekhst,al                   ;host sector to seek
;
;       active host sector?
        mov al,1
        xchg al,hstact                  ;always becomes 1
        test al,al                      ;was it already?
        jz filhst                       ;fill host if not
;
;       host buffer active, same as seek buffer?
        mov al,sekdsk
        cmp al,hstdsk                   ;sekdsk = hstdsk?
        jnz nomatch
;
;       same disk, same track?
        mov ax,hsttrk
        cmp ax,sektrk                   ;host track same as seek track
        jnz nomatch
;
;       same disk, same track, same buffer?
        mov al,sekhst
        cmp al,hstsec                   ;sekhst = hstsec?
        jz match                        ;skip if match
nomatch:
```

```
        ;proper disk, but not correct sector
        mov al, hstwrt
        test al,al                      ;"dirty" buffer ?
        jz filhst                       ;no, don't need to write
        call writehst                   ;yes, clear host buff
;       (check errors here)
;
filhst:
        ;may have to fill the host buffer
        mov al,sekdsk ! mov hstdsk,al
        mov ax,sektrk ! mov hsttrk,ax
        mov al,sekhst ! mov hstsec,al
        mov al,rsflag
        test al,al                      ;need to read?
        jz filhstl
;
        call readhst                    ;yes, if 1
;       (check errors here)
;
filhstl:
        mov hstwrt,0                    ;no pending write
;
match:
        ;copy data to or from buffer depending on "readop"
        mov al,seksec                   ;mask buffer number
        and ax,secmsk                   ;least signif bits are masked
        mov cl, 7 ! shl ax,cl           ;shift left 7 (* 128 = 2**7)
;
;       ax has relative host buffer offset
;
        add ax,offset hstbuf            ;ax has buffer address
        mov si,ax                       ;put in source index register
        mov di,dma_off                  ;user buffer is dest if readop
;
        push DS ! push ES               ;save segment registers
;
        mov ES,dma_seg                  ;set destseg to the users seg
                                        ;SI/DI and DS/ES is swapped
                                        ;if write op
        mov cx,128/2                    ;length of move in words
        mov al,readop
        test al,al                      ;which way?
        jnz     rwmove                  ;skip if read
;
;       write operation, mark and switch direction
        mov hstwrt,1                    ;hstwrt = 1 (dirty buffer now)
        xchg si,di                      ;source/dest index swap
        mov ax,DS
        mov ES,ax
        mov DS,dma_seg                  ;setup DS,ES for write
;
rwmove:
        cld ! rep movs AX,AX            ;move as 16 bit words
        pop ES ! pop DS                 ;restore segment registers
;
;       data has been moved to/from host buffer
        cmp wrtype,wrdir                ;write type to directory?
        mov al,erflag                   ;in case of errors
        jnz return_rw                   ;no further processing
;
```

```
;       clear host buffer for directory write
        test al,al                  ;errors?
        jnz return_rw               ;skip if so
        mov hstwrt,0                ;buffer written
        call writehst
        mov al,erflag
return_rw:
        ret
;
;*****************************************************
;*                                                   *
;* WRITEHST performs the physical write to the host  *
;* disk, while READHST reads the physical disk.      *
;*                                                   *
;*****************************************************
writehst:
        ret
;
readhst:
        ret
;
;*****************************************************
;*                                                   *
;* Use the GENDEF utility to create disk def tables  *
;*                                                   *
;*****************************************************
dpbase  equu    offset $
;       disk parameter tables go here
;
;*****************************************************
;*                                                   *
;* Uninitialized RAM areas follow, including the     *
;* areas created by the GENDEF utility listed above. *
;*                                                   *
;*****************************************************
sek_dsk rb      1               ;seek disk number
sek_trk rw      1               ;seek track number
sek_sec rb      1               ;seek sector number
;
hst_dsk rb      1               ;host disk number
hst_trk rw      1               ;host track number
hst_sec rb      1               ;host sector number
;
sek_hst rb      1               ;seek shr secshf
hst_act rb      1               ;host active flag
hst_wrt rb      1               ;host written flag
;
una_cnt rb      1               ;unalloc rec cnt
una_dsk rb      1               ;last unalloc disk
una_trk rw      1               ;last unalloc track
una_sec rb      1               ;last unalloc sector
;
erflag  rb      1               ;error reporting
rsflag  rb      1               ;read sector flag
readop  rb      1               ;1 if read operation
wrtype  rb      1               ;write operation type
dma_seg rw      1               ;last dma segment
dma_off rw      1               ;last dma offset
hstbuf  rb      hstsiz          ;host buffer
        end
```

```
sek_hst rb      1
hst_act rb
hst_wrt rb

una_cnt rb      1
una_dsk rb      1
una_trk rw      1
una_sec rb      1
;
erflag
rsflag
readop
wrtype
dma_seg
dma_off
hstbuf
        end
```

付録B
ランダム・アクセス・プログラムの例

この付録では，ランダム・アクセスを行なうやや幅広い完全なプログラムを紹介する．ここで示すプログラムでは，ターミナルからのコマンドにより，単にランダム・レコードの読出しおよび書込みを行なう．プログラムを作成し，アセンブルした後RANDOM．CMDという名前のファイルに格納した状態で，CCPレベルでつぎのコマンドをタイプインするとテスト・プログラムの実行を開始する．

RANDOM　X．DAT

プログラムは（この場合）X．DATという名前のファイルを探し，ファイルが見つかったとき，次の表示をしコンソールからの入力を要求する．ファイルが見つからなかったときには，ファイルを生成して同じ入力要求を表示する．

Next　Command ?

オペレータがつぎの形式をしたコマンドを入力してキャリッジ・リターン（復帰）を押す．

nW　nR　Q

ここでnは0から65535の範囲の整数値で，W，R，Qはそれぞれラムダムな書込み，ランダムな読出しおよび処理の中止を表わしている．ランダムな書込みWを指示したときには，RANDOMプログラムはつぎのように表示して入力データを要求してくる．

type　data：

それから，オペレータが127桁以内の文字をタイプインしてキャリッジ・リターン（復帰）を押すと，RANDOMプログラムの働きにより，X．DATファイルのレコード番号nに入力した文字列を書込む．ランダムな読出しRを指示したときには，RANDOMプログラムはX．DATファイルのレコード番号nのレコードを読出し，コンソール上に文字列として表示する．処理の中止Qを指示したときには，X．DATファイルをクローズし，コンソール・コマンド・プロセッサのレベルにコントロールを戻す．またどんなエラーに対しても，つぎのエラー・メッセージを表示する．

error，try　again

プログラムはファイルをオープンしたり生成する初期設定セクションから始まり，つぎにコマンドの解釈と実行を行う〝ready〟というループへと続く．またディスク処理においてはいつもオフセット005CHの標準のファイル・コントロール・ブロックおよびオフセット0080Hのバッファが用いられ，その後にユーティリティ・サブルーチンが続く．そこには〝readc.〟と呼ぶ入力を行単位に処理するプログラムが含まれている．このプログラムはランダム・アクセス処理の原理を示しているので，より高度なプログラムを開発するときの基本となる．事実，ある仕事でこのプログラムが，簡単なデータベース・マネージメント・システムに展開されている．

例えば，レコードが任意のフィールドをもつ128バイトの標準レコードの大きさであるとする．GETKEYと呼ぶプログラムは，まずシーケエンシャル・ファイルを読んで，オペレータが指定したあるフィールドを抽出する．例えば，次のコマンド，

GET　KEY　NAMES．DAT　LASTNAME　10　20

は，データベース・ファイルNAMES．DATを読み，各レコードから〝LASTNAME〟フィールド（位置10から開始し文字20で終る）を抽出する処理をGETKEYは行なう．GETKEYはそのファイル内の16ビット・レコード番号ロケーションと一緒の各LASTNAMEフィールドより構成されるテーブルをメモリに作成する．それから，GETKEYプログラムはこのリストのソートを行い，LASTNAME．KEYと呼ぶ新しいファイルを出力する．このファイル（LASTNAME．KEY）は，対応するレコード番号を持つLASTNAMEフィールドのアルファベット順のリストである（このリストは，情報検索で言えば〝インバーテッド・インデックス〟と呼ばれる）．

以上示したプログラムをQueryとリネームし，メモリにソートされたキー・ファイルを読込むために若干，改良する． コマンド行は次のようになる．

QUERY NAMES. DAT LASTNAME. KEY

番号の代りにQUERYプログラムは，NAMES. DATデータベースを検索する特別なキーである変数字文字列を読む．LASTNAME. KEY リストはソートされているので，電話帳で名前をさがすのと同じ様に〝バイナリー・サーチ″を行うことによって，じん速にあるエントリーを見つけることができる．即ち，リストの両端から開始して，両端の真中のエントリーを調べて一致していないと，次回のサーチを上の半分か，下の半分のどちらかに分ける．このようにして，探がしているアイテムへじん速に達することができる（log 2 (n)ステップで)，このアイテムから対応するレコード番号を見つけ，このレコードを取ってきてコンソールに表示する．以上の様な処理プログラムで実行することができる．

この時点で，ちょうど最初に戻っている．もう少しの労力で，上で示した128バイトレコードと異なる固定グループ・サイズも処理できる．これは，レコードないのバイト・オフセットと同様にレコード番号のトラックを保持することによって可能である．グループ・サイズを知ると，グループ・サイズが使い尽されるまで連続的に読むレコード内のグループの最初へのオフセットである適当なグループを含むレコードをランダムにアクセスする．

最後に，HARDYとLAUREL間のLASTNAME，AGEが45以下というような幾つかの関係を満足するレコードのセットを計算する計算式でQUERYを改造することができる．この条件に合ったレコードの全てを表示する．最後に，リストがメモリにとって非常に大きすぎるなら，ディスクからキー・ファイルをランダム・アクセスの機能を用いて読み出せばよい．

```
;
;*****************************************************
;*                                                   *
;*     Sample Random Access Program for CP/M-86      *
;*                                                   *
;*****************************************************
;
;    BDOS Functions
;
coninp  equ     1         ;console input function
conout  equ     2         ;console output function
pstring equ     9         ;print string until ´$´
rstring equ     10        ;read console buffer
version equ     12        ;return version number
openf   equ     15        ;file open function
closef  equ     16        ;close function
makef   equ     22        ;make file function
readr   equ     33        ;read random
writer  equ     34        ;write random
;
;   Equates for non graphic characters
cr      equ     0dh       ;carriage return
lf      equ     0ah       ;line feed
;
;
;   load SP, ready file for random access
;
        cseg
        pushf                         ;push flags in CCP stack
        pop     ax                    ;save flags in AX
        cli                           ;disable interrupts
        mov     bx,ds                 ;set SS register to base
        mov     ss,bx                 ;set SS, SP with interru
        mov     sp,offset stack       ;  for 80888
```

```
        push    ax                      ;restore the flags
        popf
;
;       CP/M-86 initial release returns the file
;       system version number of 2.2:  check is
;       shown below for illustration purposes.
;
        mov     cl,version
        call    bdos
        cmp     al,20h                  ;version 2.0 or later?
        jnb     versok
        ;       bad version, message and go back
        mov     dx,offset badver
        call    print
        jmp     abort
;
versok:
;       correct version for random access
        mov     cl,openf                ;open default fct
        mov     dx,offset fcb
        call    bdos
        inc     al                      ;err 255 becomes zero
        jnz     ready
;
;       cannot open file, so create it
        mov     cl,makef
        mov     dx,offset fcb
        call    bdos
        inc     al                      ;err 255 becomes zero
        jnz     ready
;
;       cannot create file, directory full
        mov     dx,offset nospace
        call    print
        jmp     abort                   ;back to ccp
;
;   loop back to "ready" after each command
;
ready:
;       file is ready for processing
;
        call    readcom                 ;read next command
        mov     ranrec,dx               ;store input record#
        mov     ranovf,0h               ;clear high byte if set
        cmp     al,'Q'                  ;quit?
        jnz     notq
;
;       quit processing, close file
        mov     cl,closef
        mov     dx,offset fcb
        call    bdos
        inc     al      ;err 255 becomes 0
        jz      error   ;error message, retry
        jmps    abort   ;back to ccp
;
;
;   end of quit command, process write
```

```
;
notq:
;         not the quit command, random write?
          cmp     al,'W'
          jnz     notw
;
;         this is a random write, fill buffer until cr
          mov     dx,offset datmsg
          call    print               ;data prompt
          mov     cx,127              ;up to 127 characters
          mov     bx,offset buff      ;destination
rloop:    ;read next character to buff
          push    cx                  ;save loop conntrol
          push    bx                  ;next destination
          call    getchr              ;character to AL
          pop     bx                  ;restore destination
          pop     cx                  ;restore counter
          cmp     al,cr               ;end of line?
          jz      erloop
;         not end, store character
          mov     byte ptr [bx],al
          inc     bx                  ;next to fill
          loop    rloop               ;decrement cx ..loop if
erloop:
;         end of read loop, store 00
          mov     byte ptr [bx],0h
;
;         write the record to selected record number
          mov     cl,writer
          mov     dx,offset fcb
          call    bdos
          or      al,al               ;error code zero?
          jz      ready     ;for another record
          jmps    error     ;message if not
;
;
;
;   end of write command, process read
;
;
notw:
;         not a write command, read record?
          cmp     al,'R'
          jz      ranread
          jmps    error     ;skip if not
;
;         read random record
ranread:
          mov     cl,readr
          mov     dx,offset fcb
          call    bdos
          or      al,al               ;return code 00?
          jz      readok
          jmps    error
;
;         read was successful, write to console
readok:
```

```
        call    crlf                    ;new line
        mov     cx,128                  ;max 128 characters
        mov     si,offset buff          ;next to get
wloop:
        lods    al                      ;next character
        and     al,07fh                 ;mask parity
        jnz     wloopl
        jmp     ready                   ;for another command if
wloopl:
        push    cx                      ;save counter
        push    si                      ;save next to get
        cmp     al,' '                  ;graphic?
        jb      skipw                   ;skip output if not grap
        call    putchr                  ;output character
skipw:
        pop     si
        pop     cx
        loop    wloop                   ;decrement CX and check
        jmp     ready
;
;
;   end of read command, all errors end-up here
;
;
error:
        mov     dx,offset errmsg
        call    print
        jmp     ready
;
;   BDOS entry subroutine
bdos:
        int     224                     ;entry to BDOS if by INT
        ret
;
abort:                                  ;return to CCP
        mov     cl,0
        call    bdos                    ;use function 0 to end e
;
;   utility subroutines for console i/o
;
getchr:
        ;read next console character to a
        mov     cl,coninp
        call    bdos
        ret
;
putchr:
        ;write character from a to console
        mov     cl,conout
        mov     dl,al                   ;character to send
        call    bdos                    ;send character
        ret
;
crlf:
        ;send carriage return line feed
        mov     al,cr                   ;carriage return
        call    putchr
```

```
        mov     al,lf               ;line feed
        call    putchr
        ret
;
print:
        ;print the buffer addressed by dx until $
        push    dx
        call    crlf
        pop     dx                  ;new line
        mov     cl,pstring
        call    bdos                ;print the string
        ret
;
readcom:
        ;read the next command line to the conbuf
        mov     dx,offset prompt
        call    print               ;command?
        mov     cl,rstring
        mov     dx,offset conbuf
        call    bdos                ;read command line
;       command line is present, scan it
        mov     ax,0                ;start with 0000
        mov     bx,offset conlin
readc:  mov     dl,[bx]             ;next command character
        inc     bx                  ;to next command positio
        mov     dh,0                ;zero high byte for add
        or      dl,dl               ;check for end of comman
        jnz     getnum
        ret
;       not zero, numeric?
getnum:
        sub     dl,'0'
        cmp     dl,10               ;carry if numeric
        jnb     endrd
        mov     cl,10
        mul     cl                  ;multipy accumulator by
        add     ax,dx               ;+digit
        jmps     readc              ;for another char
endrd:
;       end of read, restore value in a and return value
        mov     dx,ax               ;return value  in DX
        mov     al,-1[bx]
        cmp     al,'a'              ;check for lower case
        jnb     transl
        ret
transl: and     al,5fH  ;translate to upper case
        ret
;
;
; Template for Page 0 of Data Group
;   Contains default FCB and DMA buffer
;
        dseg
        org     05ch
fcb     rb      33                  ;default file control bl
ranrec  rw      1                   ;random record position
ranovf  rb      1                   ;high order (overflow) b
```

```
buff    rb      128                 ;default DMA buffer
;
;  string data area for console messages
badver          db      'sorry, you need cp/m version 2$'
nospace         db      'no directory space$'
datmsg          db      'type data: $'
errmsg          db      'error, try again.$'
prompt          db      'next command? $'
;
;
;    fixed and variable data area
;
conbuf db       conlen  ;length of console buffer
consiz rs       1       ;resulting size after read
conlin rs       32      ;length 32 buffer
conlen equ      offset $ - offset consiz
;
        rs      31      ;16 level stack
stack   rb      1
        db      0       ;end byte for GENCMD
        end
```

付録C

BOOT Rom

```
*****************************************************************
*                                                               *
* This is the original BOOT ROM distributed with CP/M          *
* for the SBC 86/12 and 204 Controller.  The listing           *
* is truncated on the right, but can be reproduced by          *
* assembling ROM.A86 from the distribution disk.  Note         *
* that the distributed source file should always be            *
* referenced for the latest version                            *
*                                                               *
*****************************************************************
```

```
;
; ROM bootstrap for CP/M-86 on an iSBC86/12
;               with the
;   Intel SBC 204 Floppy Disk Controller
;
;       Copyright (C) 1980,1981
;       Digital Research, Inc.
;       Box 579, Pacific Grove
;       California, 93950
;
;*********************************************
;* This is the BOOT ROM which is initiated  *
;* by a system reset.  First, the ROM moves *
;* a copy of its data area to RAM at loca-  *
;* tion 00000H, then initializes the segment*
;* registers and the stack pointer.  The    *
;* various peripheral interface chips on the*
;* SBC 86/12 are initialized.  The 8251     *
;* serial interface is configured for a 9600*
;* baud asynchronous terminal, and the in-  *
;* terrupt controller is setup for inter-   *
;* rupts 10H-17H (vectors at 00040H-0005FH) *
;* and edge-triggered auto-EOI (end of in-  *
;* terrupt) mode with all interrupt levels  *
;* masked-off.  Next, the SBC 204 Diskette  *
;* controller is initialized, and track 1   *
;* sector 1 is read to determine the target *
;* paragraph address for LOADER.  Finally,  *
;* the LOADER on track 0 sectors 2-26 and   *
;* track 1 sectors 1-26 is read into the    *
;* target address.  Control then transfers  *
;* to LOADER.  This program resides in two  *
;* 2716 EPROM's (2K each) at location       *
;* 0FF000H on the SBC 86/12 CPU board.  ROM *
;* 0 contains the even memory locations, and*
;* ROM 1 contains the odd addresses.  BOOT  *
;* ROM uses RAM between 00000H and 000FFH   *
;* (absolute) for a scratch area, along with*
;* the sector 1 buffer.                     *
;*********************************************
```

付録C

BOOT Rom

```
*******************************************************
*                                                     *
* This is the original BOOT ROM distributed with CP/M *
* for the SBC 86/12 and 204 Controller.  The listing  *
* is truncated on the right, but can be reproduced by *
* assembling ROM.A86 from the distribution disk.  Note *
* that the distributed source file should always be   *
* referenced for the latest version                   *
*                                                     *
*******************************************************
                    ;
                    ; ROM bootstrap for CP/M-86 on an iSBC86/12
                    ;              with the
                    ;   Intel SBC 204 Floppy Disk Controller
                    ;
                    ;      Copyright (C) 1980,1981
                    ;      Digital Research, Inc.
                    ;      Box 579, Pacific Grove
                    ;      California, 93950
                    ;
                    ;*********************************************
                    ;* This is the BOOT ROM which is initiated   *
                    ;* by a system reset.  First, the ROM moves *
                    ;* a copy of its data area to RAM at loca-   *
                    ;* tion 00000H, then initializes the segment*
                    ;* registers and the stack pointer.  The    *
                    ;* various peripheral interface chips on the*
                    ;* SBC 86/12 are initialized.  The 8251     *
                    ;* serial interface is configured for a 9600*
                    ;* baud asynchronous terminal, and the in-  *
                    ;* terrupt controller is setup for inter-   *
                    ;* rupts 10H-17H (vectors at 00040H-0005FH) *
                    ;* and edge-triggered auto-EOI (end of in-  *
                    ;* terrupt) mode with all interrupt levels  *
                    ;* masked-off.  Next, the SBC 204 Diskette  *
                    ;* controller is initialized, and track 1   *
                    ;* sector 1 is read to determine the target *
                    ;* paragraph address for LOADER.  Finally,  *
                    ;* the LOADER on track 0 sectors 2-26 and   *
                    ;* track 1 sectors 1-26 is read into the    *
                    ;* target address.  Control then transfers  *
                    ;* to LOADER.  This program resides in two  *
                    ;* 2716 EPROM's (2K each) at location       *
                    ;* 0FF000H on the SBC 86/12 CPU board.  ROM *
                    ;* 0 contains the even memory locations, and*
                    ;* ROM 1 contains the odd addresses.  BOOT  *
                    ;* ROM uses RAM between 00000H and 000FFH   *
                    ;* (absolute) for a scratch area, along with*
                    ;* the sector 1 buffer.                     *
                    ;*********************************************
```

```
00FF            true            equ     0ffh
FF00            false           equ     not true
                ;
00FF            debug           equ     true
                ;debug = true indicates bootstrap is in same roms
                ;with SBC 957 "Execution Vehicle" monitor
                ;at FE00:0 instead of FF00:0
                ;
000D            cr              equ     13
000A            lf              equ     10
                ;
                ;       disk ports and commands
                ;
00A0            base204         equ     0a0h
00A0            fdccom          equ     base204+0
00A0            fdcstat         equ     base204+0
00A1            fdcparm         equ     base204+1
00A1            fdcrslt         equ     base204+1
00A2            fdcrst          equ     base204+2
00A4            dmacadr         equ     base204+4
00A5            dmaccont        equ     base204+5
00A6            dmacscan        equ     base204+6
00A7            dmacsadr        equ     base204+7
00A8            dmacmode        equ     base204+8
00A8            dmacstat        equ     base204+8
00A9            fdcsel          equ     base204+9
00AA            fdcsegment      equ     base204+10
00AF            reset204        equ     base204+15
                ;
                ;actual console baud rate
2580            baud_rate       equ     9600
                ;value for 8253 baud counter
0008            baud            equ     768/(baud_rate/100)
                ;
00DA            csts            equ     0DAh    ;i8251 status port
00D8            cdata           equ     0D8h    ; "    data port
                ;
00D0            tch0            equ     0D0h    ;8253 PIC channel 0
00D2            tch1            equ     tch0+2  ;ch 1 port
00D4            tch2            equ     tch0+4  ;ch 2 port
00D6            tcmd            equ     tch0+6  ;8253 command port
                ;
00C0            icp1            equ     0C0h    ;8259a port 0
00C2            icp2            equ     0C2h    ;8259a port 1
                ;
                ;
                        IF NOT DEBUG
                ROMSEG          EQU     0FF00H  ;normal
                        ENDIF
                ;
                        IF DEBUG                ;share prom with SB
FE00            ROMSEG          EQU     0FE00H
                        ENDIF
                ;
                ;
```

```
                     ;   This long jump prom'd in by hand
                     ;       cseg      0ffffh              ;reset goes to here
                     ;       JMPF      BOTTOM              ;boot is at bottom
                     ;       EA 00 00 00 FF                ;cs = bottom of pro
                     ;                                             ip = 0
                     ;       EVEN PROM          ODD PROM
                     ;       7F8 - EA           7F8 - 00
                     ;       7F9 - 00           7F9 - 00
                     ;       7FA - FF                      ;this is not done i
                     ;
 FE00                        cseg     romseg
                     ;
                     ;First, move our data area into RAM at 0000:0200
                     ;
0000 8CC8                    mov ax,cs
0002 8ED8                    mov ds,ax           ;point DS to CS for source
0004 BE3F01                  mov SI,drombegin             ;start of data
0007 BF0002                  mov DI,offset ram_start ;offset of destinat
000A B80000                  mov ax,0
000D 8EC0                    mov es,ax           ;destination segment is 000
000F B9E600                  mov CX,data_length           ;how much to move i
0012 F3A4                    rep movs al,al               ;move out of eprom
                     ;
0014 B80000                  mov ax,0
0017 8ED8                    mov ds,ax           ;data segment now in RAM
0019 8ED0                    mov ss,ax
001B BC2A03                  mov sp,stack_offset          ;Initialize stack s
001E FC                      cld                          ;clear the directio
                     ;
                              IF NOT DEBUG
                     ;
                     ;Now, initialize the console USART and baud rate
                     ;
                              mov al,0Eh
                              out csts,al        ;give 8251 dummy mode
                              mov al,40h
                              out csts,al        ;reset 8251 to accept mode
                              mov al,4Eh
                              out csts,al        ;normal 8 bit asynch mode,
                              mov al,37h
                              out csts,al        ;enable Tx & Rx
                              mov al,0B6h
                              out tcmd,al        ;8253 ch.2 square wave mode
                              mov ax,baud
                              out tch2,al        ;low of the baud rate
                              mov al,ah
                              out tch2,al        ;high of the baud rate
                     ;
                              ENDIF
                     ;
                     ;Setup the 8259 Programmable Interrupt Controller
                     ;
001F B013                     mov al,13h
0021 E6C0                     out icp1,al        ;8259a ICW 1  8086 mode
0023 B010                     mov al,10h
```

```
0025 E6C2                out icp2,al      ;8259a ICW 2  vector @ 40-5
0027 B01F                mov al,1Fh
0029 E6C2                out icp2,al      ;8259a ICW 4  auto EOI mast
002B B0FF                mov al,0FFh
002D E6C2                out icp2,al      ;8259a OCW 1  mask all leve
                    ;
                    ;Reset and initialize the iSBC 204 Diskette Interfa
                    ;
                    restart:        ;also come back here on fatal error
002F E6AF                out reset204,AL ;reset iSBC 204 logic and
0031 B001                mov AL,1
0033 E6A2                out fdcrst,AL   ;give 8271 FDC
0035 B000                mov al,0
0037 E6A2                out fdcrst,AL   ; a reset command
0039 BB1502              mov BX,offset specs1
003C E8E100              CALL sendcom    ;program
003F BB1B02              mov BX,offset specs2
0042 E8DB00              CALL sendcom    ; Shugart SA-800 drive
0045 BB2102              mov BX,offset specs3
0048 E8D500              call sendcom    ;   characteristics
004B BB1002         homer: mov BX,offset home
004E E85800              CALL execute    ;home drive 0
                    ;
0051 BB2A03              mov bx,sector1  ;offset for first sector DM
0054 B80000              mov ax,0
0057 8EC0                mov es,ax       ;segment "      "      "    "
0059 E8A700              call setup_dma
                    ;
005C BB0202              mov bx,offset read0
005F E84700              call execute    ;get T0 S1
                    ;
0062 8E062D03            mov es,ABS
0066 BB0000              mov bx,0        ;get loader load address
0069 E89700              call setup_dma  ;setup DMA to read loader
                    ;
006C BB0602              mov bx,offset read1
006F E83700              call execute    ;read track 0
0072 BB0B02              mov bx,offset read2
0075 E83100              call execute    ;read track 1
                    ;
0078 8C06E802            mov leap_segment,ES
                    ;    setup far jump vector
007C C706E6020000        mov leap_offset,0
                    ;
                    ;    enter LOADER
0082 FF2EE602            jmpf dword ptr leap_offset
                    ;
                    pmsg:
0086 8A0F                mov cl,[BX]
0088 84C9                test cl,cl
008A 7476                jz return
008C E80400              call conout
008F 43                  inc BX
0090 E9F3FF              jmp pmsg
                    ;
```

```
                        conout:
0093 E4DA                       in al,csts
0095 A801                       test al,1
0097 74FA                       jz conout
0099 8AC1                       mov al,cl
009B E6D8                       out cdata,al
009D C3                         ret
                        ;
                        conin:
009E E4DA                       in al,csts
00A0 A802                       test al,2
00A2 74FA                       jz conin
00A4 E4D8                       in al,cdata
00A6 247F                       and al,7Fh
00A8 C3                         ret
                        ;
                        ;
                        ;
                        execute:                ;execute command string @ [BX]
                                                ;<BX> points to length,
                                                ;followed by Command byte
                                                ;followed by length-1 parameter byt
                        ;
00A9 891E0002                   mov     lastcom,BX              ;remember what it w
                        retry:                                  ;retry if not ready
00AD E87000                     call    sendcom                 ;execute the comman
                                                                ;now, let's see wha
                                                                ;of status poll was
                                                                ;for that command t
00B0 8B1E0002                   mov     BX,lastcom              ;point to command s
00B4 8A4701                     mov     AL,1[BX]                ;get command op cod
00B7 243F                       and     AL,3fh                  ;drop drive code bi
00B9 B90008                     mov     CX,0800h                ;mask if it will be
00BC 3C2C                       cmp     AL,2ch                  ;see if interrupt t
00BE 720B                       jb      execpoll
00C0 B98080                     mov     CX,8080h                ;else we use "not c
00C3 240F                       and     AL,0fh                  ;unless . . .
00C5 3C0C                       cmp     AL,0ch                  ;there isn't
00C7 B000                       mov AL,0
00C9 7737                       ja return                       ;any result at all
                        ;
                        execpoll:               ;poll for bit in b, toggled with c
00CB E4A0                       in AL,FDCSTAT
00CD 22C5                       and AL,CH
00CF 32C174F8                   xor AL,CL ! JZ execpoll
                        ;
00D3 E4A1                       in      AL,fdcrslt              ;get result registe
00D5 241E                       and     AL,1eh                  ;look only at resul
00D7 7429                       jz      return                  ;zero means it was
                        ;
00D9 3C10                       cmp al,10h
00DB 7513                       jne fatal                       ;if other than "Not
                        ;
00DD BB1302                     mov bx,offset rdstat
00E0 E83D00                     call sendcom                    ;perform read statu
```

```
                    rd_poll:
00E3 E4A0                   in al,fdc_stat
00E5 A880                   test al,80h              ;wait for command n
00E7 75FA                   jnz rd_poll
00E9 8B1E0002               mov bx,last_com          ;recover last attem
00ED E9BDFF                 jmp retry                ;and try it over ag
                    ;
                    fatal:                           ; fatal error
00F0 B400                   mov ah,0
00F2 8BD8                   mov bx,ax                ;make 16 bits
00F4 8B9F2702               mov bx,errtbl[BX]
                    ;       print appropriate error message
00F8 E88BFF                 call pmsg
00FB E8A0FF                 call conin               ;wait for key strik
00FE 58                     pop ax                   ;discard unused ite
00FF E92DFF                 jmp restart              ;then start all ove
                    ;
                    return:
0102 C3                     RET                      ;return from EXECUT
                    ;
                    setupdma:
0103 B004                   mov AL,04h
0105 E6A8                   out dmacmode,AL          ;enable dmac
0107 B000                   mov al,0
0109 E6A5                   out dmaccont,AL          ;set first (dummy)
010B B040                   mov AL,40h
010D E6A5                   out dmaccont,AL          ;force read data mo
010F 8CC0                   mov AX,ES
0111 E6AA                   out fdcsegment,AL
0113 8AC4                   mov AL,AH
0115 E6AA                   out fdcsegment,AL
0117 8BC3                   mov AX,BX
0119 E6A4                   out dmacadr,AL
011B 8AC4                   mov AL,AH
011D E6A4                   out dmacadr,AL
011F C3                     RET
                    ;
                    ;
                    ;
                    sendcom:            ;routine to send a command string t
0120 E4A0                   in AL,fdcstat
0122 2480                   and AL,80h
0124 75FA                   jnz sendcom      ;insure command not busy
0126 8A0F                   mov CL,[BX]      ;get count
0128 43                     inc BX
0129 8A07                   mov al,[BX]      ;point to and fetch command
012B E6A0                   out fdccom,AL    ;send command
                    parmloop:
012D FEC9                   dec CL
012F 74D1                   jz return        ;see if any (more) paramete
0131 43                     inc BX           ;point to next parameter
                    parmpoll:
0132 E4A0                   in AL,fdcstat
0134 2420                   and AL,20h
0136 75FA                   jnz parmpoll     ;loop until parm not full
```

```
0138 8A07                 mov AL,[BX]
013A E6A1                 out fdcparm,AL   ;output next parameter
013C E9EEFF               jmp parmloop     ;go see about another
                     ;
                     ;
                     ;    Image of data to be moved to RAM
                     ;
  013F               drombegin equ offset $
                     ;
013F 0000            clastcom        dw     0000h     ;last command
                     ;
0141 03              creadstring     db     3         ;length
0142 52                              db     52h       ;read function code
0143 00                              db     0         ;track #
0144 01                              db     1         ;sector #
                     ;
0145 04              creadtrk0       db     4
0146 53                              db     53h       ;read multiple
0147 00                              db     0         ;track 0
0148 02                              db     2         ;sectors 2
0149 19                              db     25        ;through 26
                     ;
014A 04              creadtrk1       db     4
014B 53                              db     53h
014C 01                              db     1         ;track 1
014D 01                              db     1         ;sectors 1
014E 1A                              db     26        ;through 26
                     ;
014F 026900          chome0          db     2,69h,0
0152 016C            crdstat0        db     1,6ch
0154 05350D          cspecs1         db     5,35h,0dh
0157 0808E9                          db     08h,08h,0e9h
015A 053510          cspecs2         db     5,35h,10h
015D FFFFFF                          db     255,255,255
0160 053518          cspecs3         db     5,35h,18h
0163 FFFFFF                          db     255,255,255
                     ;
0166 4702            cerrtbl dw      offset er0
0168 4702                    dw      offset er1
016A 4702                    dw      offset er2
016C 4702                    dw      offset er3
016E 5702                    dw      offset er4
0170 6502                    dw      offset er5
0172 7002                    dw      offset er6
0174 7F02                    dw      offset er7
0176 9002                    dw      offset er8
0178 A202                    dw      offset er9
017A B202                    dw      offset erA
017C C502                    dw      offset erB
017E D302                    dw      offset erC
0180 4702                    dw      offset erD
0182 4702                    dw      offset erE
0184 4702                    dw      offset erF
                     ;
0186 0D0A4E756C6C Cer0       db      cr,lf,'Null Error ??',0
```

```
     204572726F72
     203F3F00
 0186            Cer1     equ     cer0
 0186            Cer2     equ     cer0
 0186            Cer3     equ     cer0
0196 0D0A436C6F63 Cer4     db      cr,lf,'Clock Error',0
     6B204572726F
     7200
01A4 0D0A4C617465 Cer5     db      cr,lf,'Late DMA',0
     20444D4100
01AF 0D0A49442043 Cer6     db      cr,lf,'ID CRC Error',0
     524320457272
     6F7200
01BE 0D0A44617461 Cer7     db      cr,lf,'Data CRC Error',0
     204352432045
     72726F7200
01CF 0D0A44726976 Cer8     db      cr,lf,'Drive Not Ready',0
     65204E6F7420
     526561647900
01E1 0D0A57726974 Cer9     db      cr,lf,'Write Protect',0
     652050726F74
     65637400
01F1 0D0A54726B20 CerA     db      cr,lf,'Trk 00 Not Found',0
     3030204E6F74
     20466F756E64
     00
0204 0D0A57726974 CerB     db      cr,lf,'Write Fault',0
     65204661756C
     7400
0212 0D0A53656374 CerC     db      cr,lf,'Sector Not Found',0
     6F72204E6F74
     20466F756E64
     00
 0186            CerD     equ     cer0
 0186            CerE     equ     cer0
 0186            CerF     equ     cer0
                 ;
 0225            dromend equ offset $
                 ;
 00E6            data_length        equ dromend-drombegin
                 ;
                 ;         reserve space in RAM for data area
                 ;         (no hex records generated here)
                 ;
 0000                      dseg    0
                           org     0200h
                 ;
 0200            ram_start          equ     $
0200             lastcom            rw      1          ;last command
0202             read0              rb      4          ;read track 0 secto
0206             read1              rb      5          ;read T0 S2-26
020B             read2              rb      5          ;read T1 S1-26
0210             home               rb      3          ;home drive 0
0213             rdstat             rb      2          ;read status
0215             specs1             rb      6
```

```
021B            specs2          rb      6
0221            specs3          rb      6
0227            errtbl          rw      16
0247            er0             rb      length cer0     ;16
  0247          er1             equ     er0
  0247          er2             equ     er0
  0247          er3             equ     er0
0257            er4             rb      length cer4     ;14
0265            er5             rb      length cer5     ;11
0270            er6             rb      length cer6     ;15
027F            er7             rb      length cer7     ;17
0290            er8             rb      length cer8     ;18
02A2            er9             rb      length cer9     ;16
02B2            erA             rb      length cerA     ;19
02C5            erB             rb      length cerB     ;14
02D3            erC             rb      length cerC     ;19
  0247          erD             equ     er0
  0247          erE             equ     er0
  0247          erF             equ     er0
                ;
02E6            leap_offset     rw      1
02E8            leap_segment    rw      1
                ;
                ;
02EA                            rw      32      ;local stack
  032A          stack_offset    equ     offset $;stack from here do
                ;
                ;               T0 S1 read in here
  032A          sector1         equ offset $
                ;
032A            Ty              rb      1
032B            Len             rw      1
032D            Abs             rw      1       ;ABS is all we care
032F            Min             rw      1
0331            Max             rw      1
                                end
```

付録D

LDBIOS

```
*****************************************************
*                                                   *
* This is the LOADER BIOS, derived from the BIOS    *
* program by enabling the "loader_bios" condi-      *
* tional assembly switch.  The listing has been     *
* edited to remove portions which are duplicated    *
* in the BIOS listing which appears in Appendix D   *
* where elipses "..." denote the deleted portions   *
* (the listing is truncated on the right, but can   *
* be reproduced by assembling the BIOS.A86 file     *
* provided with CP/M-86)                            *
*                                                   *
*****************************************************

                  ;*********************************************
                  ;*                                           *
                  ;* Basic Input/Output System (BIOS) for      *
                  ;* CP/M-86 Configured for iSBC 86/12 with    *
                  ;* the iSBC 204 Floppy Disk Controller       *
                  ;*                                           *
                  ;* (Note: this file contains both embedded   *
                  ;* tabs and blanks to minimize the list file *
                  ;* width for printing purposes.  You may wish*
                  ;* to expand the blanks before performing    *
                  ;* major editing.)                           *
                  ;*********************************************

                  ;     Copyright (C) 1980,1981
                  ;     Digital Research, Inc.
                  ;     Box 579, Pacific Grove
                  ;     California, 93950
                  ;
                  ;     (Permission is hereby granted to use
                  ;     or abstract the following program in
                  ;     the implementation of CP/M, MP/M or
                  ;     CP/NET for the 8086 or 8088 Micro-
                  ;     processor)

 FFFF             true              equ -1
 0000             false             equ not true
```

```
                ;**********************************************
                ;*                                            *
                ;* Loader_bios is true if assembling the      *
                ;* LOADER_BIOS, otherwise BIOS is for the     *
                ;* CPM.SYS file.  Blc_list is true if we      *
                ;* have a serial printer attached to BLC8538 *
                ;* Bdos_int is interrupt used for earlier     *
                ;* versions.                                  *
                ;*                                            *
                ;**********************************************

FFFF            loader_bios         equ true
FFFF            blc_list            equ true
00E0            bdos_int            equ 224 ;reserved BDOS Interrupt

                     IF        not loader_bios
                ;---------------------------------------------
                ;|                                           |
                     . . .
                ;|                                           |
                ;---------------------------------------------
                     ENDIF     ;not loader_bios

                     IF        loader_bios
                ;---------------------------------------------
                ;|                                           |
1200            bios_code           equ 1200h ;start of LDBIOS
0003            ccp_offset          equ 0003h ;base of CPMLOADER
0406            bdos_ofst           equ 0406h ;stripped BDOS entry
                ;|                                           |
                ;---------------------------------------------
                     ENDIF     ;loader_bios
                     . . .

                     cseg
                     org       ccpoffset
                ccp:
                     org       bios_code

                ;**********************************************
                ;*                                            *
                ;* BIOS Jump Vector for Individual Routines   *
                ;*                                            *
                ;**********************************************

1200 E93C00      jmp INIT           ;Enter from BOOT ROM or LOADER
1203 E96100      jmp WBOOT          ;Arrive here from BDOS call 0
                     . . .
1239 E96400      jmp GETIOBF        ;return I/O map byte (IOBYTE)
123C E96400      jmp SETIOBF        ;set I/O map byte (IOBYTE)
```

```
                  ;*********************************************
                  ;*                                           *
                  ;* INIT Entry Point, Differs for LDBIOS and  *
                  ;* BIOS, according to "Loader_Bios" value    *
                  ;*                                           *
                  ;*********************************************

                  INIT:   ;print signon message and initialize hardwa
123F 8CC8                 mov ax,cs          ;we entered with a JMPF so
1241 8ED0                 mov ss,ax          ; CS: as the initial value
1243 8ED8                 mov ds,ax          ;       DS:,
1245 8EC0                 mov es,ax          ;       and ES:
                          ;use local stack during initialization
1247 BCA916               mov sp,offset stkbase
124A FC                   cld                ;set forward direction

                          IF     not loader_bios
                  ;-------------------------------------------
                  ;|                                         |
                          ; This is a BIOS for the CPM.SYS file.
                  ;|       . . .                            |
                  ;-------------------------------------------
                          ENDIF  ;not loader_bios

                          IF     loader_bios
                  ;-------------------------------------------
                  ;|                                         |
                          ;This is a BIOS for the LOADER
124B 1E                   push ds            ;save data segment
124C B80000               mov ax,0
124F 8ED8                 mov ds,ax          ;point to segment zero
                          ;BDOS interrupt offset
1251 C70680030604         mov bdos_offset,bdos_ofst
1257 8C0E8203             mov bdos_segment,CS ;bdos interrupt segment
125B 1F                   pop ds             ;restore data segment
                  ;|                                         |
                  ;-------------------------------------------
                          ENDIF  ;loader_bios

125C BB1514               mov bx,offset signon
125F E85A00               call pmsg          ;print signon message
1262 B100                 mov cl,0           ;default to dr A: on coldst
1264 E99CED               jmp ccp            ;jump to cold start entry o

1267 E99FED       WBOOT:  jmp ccp+6          ;direct entry to CCP at com

                          IF     not loader_bios
                  ;-------------------------------------------
                  ;|                                         |
                          . . .
                  ;|                                         |
                  ;-------------------------------------------
                          ENDIF  ;not loader_bios
```

```
                ;*********************************************
                ;*                                           *
                ;*   CP/M Character I/O Interface Routines   *
                ;*   Console is Usart (i8251a) on iSBC 86/12 *
                ;*   at ports D8/DA                          *
                ;*                                           *
                ;*********************************************

                CONST:          ;console status
126A E4DA               in al,csts
                        . . .
                const_ret:
1272 C3                 ret                     ;Receiver Data Available

                CONIN:                          ;console input
1273 E8F4FF             call const
                        . . .
                 CONOUT:        ;console output
127D E4DA               in al,csts
                        . . .

                LISTOUT:                        ;list device output

                        IF      blc_list
                ;---------------------------------------------
                ;|                                           |
1288 E80700             call LISTST
                          . . .
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;blc_list

1291 C3                 ret

                LISTST:                         ;poll list status

                        IF      blc_list
                ;---------------------------------------------
                ;|                                           |
1292 E441               in al,lsts
                        . . .
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;blc_list

129C C3                 ret

                PUNCH:  ;not implemented in this configuration
                READER:
129D B01A               mov al,lah
129F C3                 ret                     ;return EOF for now
```

```
                GETIOBF:
12A0 B000              mov al,0         ;TTY: for consistency
12A2 C3                ret              ;IOBYTE not implemented

                SETIOBF:
12A3 C3                ret              ;iobyte not implemented

                zero_ret:
12A4 2400              and al,0
12A6 C3         ret                     ;return zero in AL and flag

                ; Routine to get and echo a console character
                ;      and shift it to upper case

                uconecho:
12A7 E8C9FF            call CONIN       ;get a console character
                       . . .
                ;*******************************************
                ;*                                         *
                ;*         Disk Input/Output Routines      *
                ;*                                         *
                ;*******************************************

                SELDSK:      ;select disk given by register CL
12CA BB0000            mov bx,0000h
                       . . .

                HOME:  ;move selected disk to home position (Track
12EB C606311500        mov trk,0        ;set disk i/o to track zero
                       . . .

                SETTRK: ;set track address given by CX
1300 880E3115          mov trk,cl       ;we only use 8 bits of trac
1304 C3                ret

                SETSEC: ;set sector number given by cx
1305 880E3215          mov sect,cl      ;we only use 8 bits of sect
1309 C3                ret

                SECTRAN: ;translate sector CX using table at [DX]
130A 8BD9              mov bx,cx
                       . . .

                SETDMA: ;set DMA offset given by CX
1311 890E2A15          mov dma_adr,CX
1315 C3                ret

                SETDMAB: ;set DMA segment given by CX
1316 890E2C15          mov dma_seg,CX
131A C3                ret
                ;
                GETSEGT:  ;return address of physical memory table
131B BB3815            mov bx,offset seg_table
131E C3                ret
```

```
                ;*********************************************
                ;*                                           *
                ;*  All disk I/O parameters are setup:  the *
                ;*  Read and Write entry points transfer one *
                ;*  sector of 128 bytes to/from the current  *
                ;*  DMA address using the current disk drive *
                ;*                                           *
                ;*********************************************

                READ:
131F B012               mov al,12h      ;basic read sector command
1321 EB02               jmps r_w_common

                WRITE:
1323 B00A               mov al,0ah      ;basic write sector command

                r_w_common:
1325 BB2F15             mov bx,offset io_com ;point to command stri
                        . . .

                ;*********************************************
                ;*                                           *
                ;*               Data Areas                  *
                ;*                                           *
                ;*********************************************
  1415          data_offset     equ offset $

                        dseg
                        org     data_offset     ;contiguous with co

                        IF      loader_bios
                ;---------------------------------------------
                ;|                                           |
1415 0D0A0D0A   signon  db      cr,lf,cr,lf
1419 43502F4D2D38       db      'CP/M-86 Version 2.2',cr,lf,0
     362056657273
     696F6E20322E
     320D0A00
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;loader_bios

                        IF      not loader_bios
                ;---------------------------------------------
                ;|                                           |
                        . . .
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;not loader_bios

142F 0D0A486F6D65 bad_hom db    cr,lf,'Home Error',cr,lf,0
                        . . .
=                       include singles.lib ;read in disk definitio
=               ;       DISKS 2
```

```
=  1541                 dpbase   equ      $                 ;Base of Disk Param
                                 . . .
=1668 00                         db       0                 ;Marks End of Modul

 1669                   loc_stk rw  32  ;local stack for initialization
   16A9                 stkbase equ offset $
                                 . . .
 16A9 00                         db 0    ;fill last address for GENCMD

                        ;*********************************************
                        ;*                                           *
                        ;*          Dummy Data Section               *
                        ;*                                           *
                        ;*********************************************
  0000                          dseg     0          ;absolute low memory
                                org      0          ;(interrupt vectors)
                                 . . .
                                END
```

付録E

BIOS

```
*****************************************************
*                                                   *
* This is the CP/M-86 BIOS, derived from the BIOS  *
* program by disabling the "loader_bios" condi-     *
* tional assembly switch.  The listing has been     *
* truncated on the right, but can be reproduced     *
* by assembling the BIOS.A86 file provided with     *
* CP/M-86.   This BIOS allows CP/M-86 operation     *
* with the Intel SBC 86/12 with the SBC 204 con-    *
* troller.  Use this BIOS, or the skeletal CBIOS    *
* listed in Appendix E, as the basis for a cus-     *
* tomized implementation of CP/M-86.                *
* provided with CP/M-86)                            *
*                                                   *
*****************************************************
```

```
                    ;*********************************************
                    ;*                                           *
                    ;* Basic Input/Output System (BIOS) for      *
                    ;* CP/M-86 Configured for iSBC 86/12 with    *
                    ;* the iSBC 204 Floppy Disk Controller       *
                    ;*                                           *
                    ;* (Note:  this file contains both embedded  *
                    ;* tabs and blanks to minimize the list file *
                    ;* width for printing purposes.  You may wish*
                    ;* to expand the blanks before performing    *
                    ;* major editing.)                           *
                    ;*********************************************
                    ;
                    ;         Copyright (C) 1980,1981
                    ;         Digital Research, Inc.
                    ;         Box 579, Pacific Grove
                    ;         California, 93950
                    ;
                    ;         (Permission is hereby granted to use
                    ;         or abstract the following program in
                    ;         the implementation of CP/M, MP/M or
                    ;         CP/NET for the 8086 or 8088 Micro-
                    ;         processor)

 FFFF               true            equ -1
 0000               false           equ not true
```

```
                 ;*********************************************
                 ;*                                           *
                 ;* Loader_bios is true if assembling the     *
                 ;* LOADER_BIOS, otherwise BIOS is for the    *
                 ;* CPM.SYS file.  Blc_list is true if we     *
                 ;* have a serial printer attached to BLC8538 *
                 ;* Bdos_int is interrupt used for earlier    *
                 ;* versions.                                 *
                 ;*                                           *
                 ;*********************************************

0000             loader_bios         equ false
FFFF             blc_list            equ true
00E0             bdos_int            equ 224 ;reserved BDOS Interrupt

                         IF          not loader_bios
                 ;--------------------------------------------
                 ;|                                           |
2500             bios_code           equ 2500h
0000             ccp_offset          equ 0000h
0B06             bdos_ofst           equ 0B06h ;BDOS entry point
                 ;|                                           |
                 ;--------------------------------------------
                         ENDIF       ;not loader_bios

                         IF          loader_bios
                 ;--------------------------------------------
                 ;|                                           |
                 bios_code           equ 1200h ;start of LDBIOS
                 ccp_offset          equ 0003h ;base of CPMLOADER
                 bdos_ofst           equ 0406h ;stripped BDOS entry
                 ;|                                           |
                 ;--------------------------------------------
                         ENDIF       ;loader_bios

00DA             csts                equ 0DAh  ;i8251 status port
00D8             cdata               equ 0D8h  ;   "  data port

                         IF          blc_list
                 ;--------------------------------------------
                 ;|                                           |
0041             lsts                equ 41h ;2651 No. 0 on BLC8538 stat
0040             ldata               equ 40h ; "   "   "  "    "    data
0060             blc_reset           equ 60h ;reset selected USARTS on B
                 ;|                                           |
                 ;--------------------------------------------
                         ENDIF       ;blc_list

                 ;*********************************************
                 ;*                                           *
                 ;*    Intel iSBC 204 Disk Controller Ports   *
                 ;*                                           *
                 ;*********************************************
```

```
00A0            base204         equ 0a0h            ;SBC204 assigned ad

00A0            fdc_com         equ base204+0       ;8271 FDC out comma
00A0            fdc_stat        equ base204+0       ;8271 in status
00A1            fdc_parm        equ base204+1       ;8271 out parameter
00A1            fdc_rslt        equ base204+1       ;8271 in result
00A2            fdc_rst         equ base204+2       ;8271 out reset
00A4            dmac_adr        equ base204+4       ;8257 DMA base addr
00A5            dmac_cont       equ base204+5       ;8257 out control
00A6            dmac_scan       equ base204+6       ;8257 out scan cont
00A7            dmac_sadr       equ base204+7       ;8257 out scan addr
00A8            dmac_mode       equ base204+8       ;8257 out mode
00A8            dmac_stat       equ base204+8       ;8257 in status
00A9            fdc_sel         equ base204+9       ;FDC select port (n
00AA            fdc_segment     equ base204+10      ;segment address re
00AF            reset_204       equ base204+15      ;reset entire inter

000A            max_retries     equ 10              ;max retries on dis
                                                    ;before perm error
000D            cr              equ 0dh             ;carriage return
000A            lf              equ 0ah             ;line feed

                        cseg
                        org     ccpoffset
                ccp:
                        org     bios_code

                ;*********************************************
                ;*                                           *
                ;* BIOS Jump Vector for Individual Routines  *
                ;*                                           *
                ;*********************************************

2500 E93C00             jmp INIT        ;Enter from BOOT ROM or LOADER
2503 E98400             jmp WBOOT       ;Arrive here from BDOS call 0
2506 E99000             jmp CONST       ;return console keyboard status
2509 E99600             jmp CONIN       ;return console keyboard char
250C E99D00             jmp CONOUT      ;write char to console device
250F E9A500             jmp LISTOUT     ;write character to list device
2512 E9B700             jmp PUNCH       ;write character to punch device
2515 E9B400             jmp READER      ;return char from reader device
2518 E9FF00             jmp HOME        ;move to trk 00 on cur sel drive
251B E9DB00             jmp SELDSK      ;select disk for next rd/write
251E E90E01             jmp SETTRK      ;set track for next rd/write
2521 E91001             jmp SETSEC      ;set sector for next rd/write
2524 E91901             jmp SETDMA      ;set offset for user buff (DMA)
2527 E92401             jmp READ        ;read a 128 byte sector
252A E92501             jmp WRITE       ;write a 128 byte sector
252D E99100             jmp LISTST      ;return list status
2530 E90601             jmp SECTRAN     ;xlate logical->physical sector
2533 E90F01             jmp SETDMAB     ;set seg base for buff (DMA)
2536 E91101             jmp GETSEGT     ;return offset of Mem Desc Table
2539 E99300             jmp GETIOBF     ;return I/O map byte (IOBYTE)
253C E99300             jmp SETIOBF     ;set I/O map byte (IOBYTE)
```

```
                ;*****************************************************
                ;*                                                   *
                ;* INIT Entry Point, Differs for LDBIOS and          *
                ;* BIOS, according to "Loader_Bios" value            *
                ;*                                                   *
                ;*****************************************************

                INIT:   ;print signon message and initialize hardwa
253F 8CC8               mov ax,cs           ;we entered with a JMPF so
2541 8ED0               mov ss,ax           ; CS: as the initial value
2543 8ED8               mov ds,ax           ;      DS:,
2545 8EC0               mov es,ax           ;      and ES:
                        ;use local stack during initialization
2547 BCE429             mov sp,offset stkbase
254A FC                 cld                 ;set forward direction

                        IF      not loader_bios
                ;---------------------------------------------------
                ;|                                                 |
                        ; This is a BIOS for the CPM.SYS file.
                        ; Setup all interrupt vectors in low
                        ; memory to address trap

254B 1E                 push ds             ;save the DS register
254C B80000             mov ax,0
254F 8ED8               mov ds,ax
2551 8EC0               mov es,ax           ;set ES and DS to zero
                        ;setup interrupt 0 to address trap routine
2553 C70600008D25       mov int0_offset,offset int_trap
2559 8C0E0200           mov int0_segment,CS
255D BF0400             mov di,4
2560 BE0000             mov si,0            ;then propagate
2563 B9FE01             mov cx,510          ;trap vector to
2566 F3A5               rep movs ax,ax      ;all 256 interrupts
                        ;BDOS offset to proper interrupt
2568 C7068003060B       mov bdos_offset,bdos_ofst
256E 1F                 pop ds              ;restore the DS register

                ;*****************************************************
                ;*                                                   *
                ;* National "BLC 8538" Channel 0 for a serial*
                ;* 9600 baud printer - this board uses 8 Sig-*
                ;* netics 2651 Usarts which have on-chip baud*
                ;* rate generators.                                  *
                ;*                                                   *
                ;*****************************************************

256F B0FF               mov al,0FFh
2571 E660               out blc_reset,al ;reset all usarts on 8538
2573 B04E               mov al,4Eh
2575 E642               out ldata+2,al ;set usart 0 in async 8 bit
2577 B03E               mov al,3Eh
2579 E642               out ldata+2,al ;set usart 0 to 9600 baud
257B B037               mov al,37h
257D E643               out ldata+3,al  ;enable Tx/Rx, and set up R
```

```
                    ;|                                          |
                    ;--------------------------------------------
                          ENDIF   ;not loader_bios

                          IF     loader_bios
                    ;--------------------------------------------
                    ;|                                          |
                          ;This is a BIOS for the LOADER
                          push ds           ;save data segment
                          mov ax,0
                          mov ds,ax         ;point to segment zero
                          ;BDOS interrupt offset
                          mov bdos_offset,bdos_ofst
                          mov bdos_segment,CS ;bdos interrupt segment
                          pop ds            ;restore data segment
                    ;|                                          |
                    ;--------------------------------------------
                          ENDIF   ;loader_bios

257F BB4427               mov bx,offset signon
2582 E86600               call pmsg         ;print signon message
2585 B100                 mov cl,0          ;default to dr A: on coldst
2587 E976DA               jmp ccp           ;jump to cold start entry o

258A E979DA         WBOOT: jmp ccp+6        ;direct entry to CCP at com

                          IF     not loader_bios
                    ;--------------------------------------------
                    ;|                                          |
                    int_trap:
258D FA                   cli               ;block interrupts
258E 8CC8                 mov ax,cs
2590 8ED8                 mov ds,ax         ;get our data segment
2592 BB7927               mov bx,offset int_trp
2595 E85300               call pmsg
2598 F4                   hlt               ;hardstop
                    ;|                                          |
                    ;--------------------------------------------
                          ENDIF   ;not loader_bios

                    ;*********************************************
                    ;*                                           *
                    ;*   CP/M Character I/O Interface Routines   *
                    ;*   Console is Usart (i8251a) on iSBC 86/12 *
                    ;*   at ports D8/DA                          *
                    ;*                                           *
                    ;*********************************************

                    CONST:            ;console status
2599 E4DA                 in al,csts
259B 2402                 and al,2
259D 7402                 jz const_ret
259F 0CFF                 or al,255         ;return non-zero if RDA
                    const_ret:
25A1 C3                   ret               ;Receiver Data Available
```

```
                    CONIN:                  ;console input
25A2 E8F4FF                   call const
25A5 74FB                     jz CONIN      ;wait for RDA
25A7 E4D8                     in al,cdata
25A9 247F                     and al,7fh    ;read data and remove parit
25AB C3                       ret

                    CONOUT:     ;console output
25AC E4DA                     in al,csts
25AE 2401                     and al,1      ;get console status
25B0 74FA                     jz CONOUT     ;wait for TBE
25B2 8AC1                     mov al,cl
25B4 E6D8                     out cdata,al  ;Transmitter Buffer Empty
25B6 C3                       ret           ;then return data

                    LISTOUT:                ;list device output

                              IF      blc_list
                    ;---------------------------------------------
                    ;|                                           |
25B7 E80700                   call LISTST
25BA 74FB                     jz LISTOUT    ;wait for printer not busy
25BC 8AC1                     mov al,cl
25BE E640                     out ldata,al  ;send char to TI 810
                    ;|                                           |
                    ;---------------------------------------------
                              ENDIF  ;blc_list

25C0 C3                       ret

                    LISTST:                 ;poll list status

                              IF      blc_list
                    ;---------------------------------------------
                    ;|                                           |
25C1 E441                     in al,lsts
25C3 2481                     and al,81h    ;look at both TxRDY and DTR
25C5 3C81                     cmp al,81h
25C7 750A                     jnz zero_ret  ;either false, printer is b
25C9 0CFF                     or al,255     ;both true, LPT is ready
                    ;|                                           |
                    ;---------------------------------------------
                              ENDIF  ;blc_list

25CB C3                       ret

                    PUNCH:  ;not implemented in this configuration
                    READER:
25CC B01A                     mov al,lah
25CE C3                       ret           ;return EOF for now

                    GETIOBF:
25CF B000                     mov al,0      ;TTY: for consistency
25D1 C3                       ret           ;IOBYTE not implemented
```

```
                    SETIOBF:
25D2 C3                     ret                 ;iobyte not implemented

                    zero_ret:
25D3 2400                   and al,0
25D5 C3             ret                         ;return zero in AL and flag

                    ; Routine to get and echo a console character
                    ;       and shift it to upper case

                    uconecho:
25D6 E8C9FF                 call CONIN          ;get a console character
25D9 50                     push ax
25DA 8AC8                   mov cl,al           ;save and
25DC E8CDFF                 call CONOUT
25DF 58                     pop ax              ;echo to console
25E0 3C61                   cmp al,'a'
25E2 7206                   jb uret             ;less than 'a' is ok
25E4 3C7A                   cmp al,'z'
25E6 7702                   ja uret             ;greater than 'z' is ok
25E8 2C20                   sub al,'a'-'A'      ;else shift to caps
                    uret:
25EA C3                     ret

                    ;       utility subroutine to print messages

                    pmsg:
25EB 8A07                   mov al,[BX]         ;get next char from message
25ED 84C0                   test al,al
25EF 7428                   jz return           ;if zero return
25F1 8AC8                   mov CL,AL
25F3 E8B6FF                 call CONOUT         ;print it
25F6 43                     inc BX
25F7 EBF2                   jmps pmsg           ;next character and loop

                    ;*******************************************
                    ;*                                         *
                    ;*          Disk Input/Output Routines     *
                    ;*                                         *
                    ;*******************************************

                    SELDSK:     ;select disk given by register CL
25F9 BB0000                 mov bx,0000h
25FC 80F902                 cmp cl,2            ;this BIOS only supports 2
25FF 7318                   jnb return          ;return w/ 0000 in BX if ba
2601 B080                   mov al, 80h
2603 80F900                 cmp cl,0
2606 7502                   jne sell            ;drive 1 if not zero
2608 B040                   mov al, 40h         ;else drive is 0
260A A26928         sell:   mov sel_mask,al     ;save drive select mask
                                                ;now, we need disk paramete
260D B500
260F 8BD9                   mov ch,0
2611 B104                   mov bx,cx           ;BX = word(CL)
                            mov cl,4
```

```
2613 D3E3                  shl bx,cl         ;multiply drive code * 16
                           ;create offset from Disk Parameter Base
2615 81C37C28              add bx,offset dp_base
                  return:
2619 C3                    ret

                  HOME:    ;move selected disk to home position (Track
261A C6066C2800            mov trk,0         ;set disk i/o to track zero
261F BB6E28                mov bx,offset hom_com
2622 E83500                call execute
2625 74F2                  jz return         ;home drive and return if O
2627 BB6A27                mov bx,offset bad_hom   ;else print
262A E8BEFF                call pmsg         ;"Home Error"
262D EBEB                  jmps home         ;and retry

                  SETTRK: ;set track address given by CX
262F 880E6C28              mov trk,cl        ;we only use 8 bits of trac
2633 C3                    ret

                  SETSEC: ;set sector number given by cx
2634 880E6D28              mov sect,cl       ;we only use 8 bits of sect
2638 C3                    ret

                  SECTRAN: ;translate sector CX using table at [DX]
2639 8BD9                  mov bx,cx
263B 03DA                  add bx,dx         ;add sector to tran table a
263D 8A1F                  mov bl,[bx]       ;get logical sector
263F C3                    ret

                  SETDMA: ;set DMA offset given by CX
2640 890E6528              mov dma_adr,CX
2644 C3                    ret

                  SETDMAB: ;set DMA segment given by CX
2645 890E6728              mov dma_seg,CX
2649 C3                    ret
                  ;
                  GETSEGT:  ;return address of physical memory table
264A BB7328                mov bx,offset seg_table
264D C3                    ret

                  ;*********************************************
                  ;*                                           *
                  ;*  All disk I/O parameters are setup:  the  *
                  ;*  Read and Write entry points transfer one *
                  ;*  sector of 128 bytes to/from the current  *
                  ;*  DMA address using the current disk drive *
                  ;*                                           *
                  ;*********************************************

                  READ:
264E B012                  mov al,12h        ;basic read sector command
2650 EB02                  jmps r_w_common

                  WRITE:
```

```
2652 B00A                   mov al,0ah              ;basic write sector command

                    r_w_common:
2654 BB6A28                 mov bx,offset io_com ;point to command stri
2657 884701                 mov byte ptr 1[BX],al ;put command into str
                    ;       fall into execute and return

                    execute:  ;execute command string.
                            ;[BX] points to length,
                            ;       followed by Command byte,
                            ;       followed by length-1 parameter byte

265A 891E6328               mov last_com,BX ;save command address for r
                    outer_retry:
                            ;allow some retrying
265E C60662280A             mov rtry_cnt,max_retries
                    retry:
2663 8B1E6328               mov BX,last_com
2667 E88900                 call send_com     ;transmit command to i8271
                    ;       check status poll

266A 8B1E6328               mov BX,last com
266E 8A4701                 mov al,1[bx]      ;get command op code
2671 B90008                 mov cx,0800h      ;mask if it will be "int re
2674 3C2C                   cmp al,2ch
2676 720B                   jb exec_poll      ;ok if it is an interrupt t
2678 B98080                 mov cx,8080h      ;else we use "not command b
267B 240F                   and al,0fh
267D 3C0C                   cmp al,0ch        ;unless there isn't
267F B000                   mov al,0
2681 7736                   ja exec_exit      ;       any result
                                              ;poll for bits in CH,
                    exec_poll:                ;  toggled with bits in CL

2683 E4A0                   in al,fdc_stat  ;read status
2685 22C5                   and al,ch
2687 32C1                   xor al,cl         ; isolate what we want to
2689 74F8                   jz exec_poll    ;and loop until it is done

                                              ;Operation complete,
268B E4A1                   in al,fdc_rslt  ; see if result code indica
268D 241E                   and al,leh
268F 7428                   jz exec_exit      ;no error, then exit
                                              ;some type of error occurre
2691 3C10                   cmp al,10h
2693 7425                   je dr_nrdy        ;was it a not ready drive ?
                                              ;no,
                    dr_rdy: ; then we just retry read or write
2695 FE0E6228               dec rtry_cnt
2699 75C8                   jnz retry         ; up to 10 times

                    ;       retries do not recover from the
                    ;       hard error

269B B400                   mov ah,0
```

```
269D 8BD8           mov bx,ax          ;make error code 16 bits
269F 8B9F9127       mov bx,errtbl[BX]
26A3 E845FF         call pmsg          ;print appropriate message
26A6 E4D8           in al,cdata        ;flush usart receiver buffe
26A8 E82BFF         call uconecho      ;read upper case console ch
26AB 3C43           cmp al,'C'
26AD 7425           je wboot_l         ;cancel
26AF 3C52           cmp al,'R'
26B1 74AB           je outer_retry     ;retry 10 more times
26B3 3C49           cmp al,'I'
26B5 741A           je z_ret           ;ignore error
26B7 0CFF           or al,255          ;set code for permanent err
               exec_exit:
26B9 C3             ret

               dr_nrdy:          ;here to wait for drive ready
26BA E81A00         call test_ready
26BD 75A4           jnz retry          ;if it's ready now we are d
26BF E81500         call test_ready
26C2 759F           jnz retry          ;if not ready twice in row,
26C4 BB0228         mov bx,offset nrdymsg
26C7 E821FF         call pmsg ;"Drive Not Ready"
               nrdy01:
26CA E80A00         call test_ready
26CD 74FB           jz nrdy01          ;now loop until drive ready
26CF EB92           jmps retry         ;then go retry without decr
               zret:
26D1 2400           and al,0
26D3 C3             ret                ;return with no error code

               wboot_l:                ;can't make it w/ a short l
26D4 E9B3FE         jmp WBOOT

               ;******************************************
               ;*                                        *
               ;*  The i8271 requires a read status command *
               ;*  to reset a drive-not-ready after the  *
               ;*  drive becomes ready                   *
               ;*                                        *
               ;******************************************

               test_ready:
26D7 B640           mov dh, 40h        ;proper mask if dr 1
26D9 F606692880     test sel_mask,80h
26DE 7502           jnz nrdy2
26E0 B604           mov dh, 04h        ;mask for dr 0 status bit
               nrdy2:
26E2 BB7128         mov bx,offset rds_com
26E5 E80B00         call send_com
               dr_poll:
26E8 E4A0           in al,fdc_stat     ;get status word
26EA A880           test al,80h
26EC 75FA           jnz dr_poll        ;wait for not command busy
26EE E4A1           in al,fdc_rslt     ;get "special result"
26F0 84C6           test al,dh         ;look at bit for this drive
```

```
26F2 C3                 ret                     ;return status of ready
                ;*******************************************
                ;*                                         *
                ;*  Send_com sends a command and parameters *
                ;*  to the i8271:  BX addresses parameters.  *
                ;*  The DMA controller is also initialized  *
                ;*  if this is a read or write              *
                ;*                                         *
                ;*******************************************

                send_com:
26F3 E4A0               in al,fdc_stat
26F5 A880               test al,80h         ;insure command not busy
26F7 75FA               jnz send_com        ;loop until ready

                        ;see if we have to initialize for a DMA ope

26F9 8A4701             mov al,1[bx]        ;get command byte
26FC 3C12               cmp al,12h
26FE 7504               jne write_maybe ;if not a read it could be
2700 B140               mov cl,40h
2702 EB06               jmps init_dma       ;is a read command, go set
                write_maybe:
2704 3C0A               cmp al,0ah
2706 7520               jne dma_exit        ;leave DMA alone if not rea
2708 B180               mov cl,80h          ;we have write, not read
                init_dma:
                ;we have a read or write operation, setup DMA contr
                ;               (CL contains proper direction bit)
270A B004               mov al,04h
270C E6A8               out dmac_mode,al        ;enable dmac
270E B000               mov al,00
2710 E6A5               out dmac_cont,al        ;send first byte to con
2712 8AC1               mov al,cl
2714 E6A5               out dmac_cont,al       ;load direction register
2716 A16528             mov ax,dma_adr
2719 E6A4               out dmac_adr,al        ;send low byte of DMA
271B 8AC4               mov al,ah
271D E6A4               out dmac_adr,al        ;send high byte
271F A16728             mov ax,dma_seg
2722 E6AA               out fdc_segment,al ;send low byte of segmen
2724 8AC4               mov al,ah
2726 E6AA               out fdc_segment,al ;then high segment addre
                dma_exit:
2728 8A0F               mov cl,[BX]         ;get count
272A 43                 inc BX
272B 8A07               mov al,[BX]         ;get command
272D 0A066928           or al,sel_mask      ;merge command and drive co
2731 E6A0               out fdc_com,al      ;send command byte
                parm_loop:
2733 FEC9               dec cl
2735 7482               jz exec_exit        ;no (more) parameters, retu
2737 43                 inc BX              ;point to (next) parameter
                parm_poll:
```

```
2738 E4A0                in al,fdc_stat
273A A820                test al,20h       ;test "parameter register f
273C 75FA                jnz parm_poll     ;idle until parm req not fu
273E 8A07                mov al,[BX]
2740 E6A1                out fdc_parm,al ;send next parameter
2742 EBEF                jmps parm_loop  ;go see if there are more p

                    ;*********************************************
                    ;*                                           *
                    ;*              Data Areas                   *
                    ;*                                           *
                    ;*********************************************
 2744               data_offset     equ offset $

                         dseg
                         org       data_offset      ;contiguous with co

                         IF        loader_bios
                    ;--------------------------------------------
                    ;|                                           |
                    signon   db    cr,lf,cr,lf
                             db    'CP/M-86 Version 2.2',cr,lf,0
                    ;|                                           |
                    ;--------------------------------------------
                         ENDIF     ;loader_bios

                         IF        not loader_bios
                    ;--------------------------------------------
                    ;|                                           |
2744 0D0A0D0A       signon   db    cr,lf,cr,lf
2748 202053797374            db    '  System Generated  - 11 Jan 81',c
     656D2047656E
     657261746564
     20202D203131
     204A616E2038
     310D0A00
                    ;|                                           |
                    ;--------------------------------------------
                         ENDIF     ;not loader_bios

276A 0D0A486F6D65 bad_hom db   cr,lf,'Home Error',cr,lf,0
     204572726F72
     0D0A00
2779 0D0A496E7465 int_trp db   cr,lf,'Interrupt Trap Halt',cr,lf,0
     727275707420
     547261702048
     616C740D0A00

2791 B127B127B127 errtbl dw    er0,er1,er2,er3
     B127
2799 C127D127DE27        dw    er4,er5,er6,er7
     EF27
27A1 022816282828        dw    er8,er9,erA,erB
     3D28
27A9 4D28B127B127        dw    erC,erD,erE,erF
```

B127

```
27B1 0D0A4E756C6C er0      db  cr,lf,'Null Error ??',0
     204572726F72
     203F3F00
 27B1                er1      equ er0
 27B1                er2      equ er0
 27B1                er3      equ er0
27C1 0D0A436C6F63 er4      db  cr,lf,'Clock Error :',0
     6B204572726F
     72203A00
27D1 0D0A4C617465 er5      db  cr,lf,'Late DMA :',0
     20444D41203A
     00
27DE 0D0A49442043 er6      db  cr,lf,'ID CRC Error :',0
     524320457272
     6F72203A00
27EF 0D0A44617461 er7      db  cr,lf,'Data CRC Error :',0
     204352432045
     72726F72203A
     00
2802 0D0A44726976 er8      db  cr,lf,'Drive Not Ready :',0
     65204E6F7420
     526561647920
     3A00
2816 0D0A57726974 er9      db  cr,lf,'Write Protect :',0
     652050726F74
     656374203A00
2828 0D0A54726B20 erA      db  cr,lf,'Trk 00 Not Found :',0
     3030204E6F74
     20466F756E64
     203A00
283D 0D0A57726974 erB      db  cr,lf,'Write Fault :',0
     65204661756C
     74203A00
284D 0D0A53656374 erC      db  cr,lf,'Sector Not Found :',0
     6F72204E6F74
     20466F756E64
     203A00
 27B1                erD      equ er0
 27B1                erE      equ er0
 27B1                erF      equ er0
 2802                nrdymsg equ er8

2862 00              rtry_cnt db 0   ;disk error retry counter
2863 0000            last_com dw 0   ;address of last command string
2865 0000            dma_adr  dw 0   ;dma offset stored here
2867 0000            dma_seg  dw 0   ;dma segment stored here
2869 40              sel_mask db 40h ;select mask, 40h or 80h

                     ;        Various command strings for i8271

286A 03              io_com   db 3   ;length
286B 00              rd_wr    db 0   ;read/write function code
286C 00              trk      db 0   ;track #
```

```
286D 00             sect    db 0    ;sector #

286E 022900         hom_com db 2,29h,0        ;home drive command
2871 012C           rds_com db 1,2ch          ;read status command

                    ;       System Memory Segment Table

2873 02             segtable db 2   ;2 segments
2874 DF02                   dw tpa_seg        ;1st seg starts after BIOS
2876 2105                   dw tpa_len        ;and extends to 08000
2878 0020                   dw 2000h          ;second is 20000 -
287A 0020                   dw 2000h          ;3FFFF (128k)

=                           include singles.lib ;read in disk definitio
=                   ;               DISKS 2
=   287C            dpbase  equ     $               ;Base of Disk Param
=287C AB280000      dpe0    dw      xlt0,0000h      ;Translate Table
=2880 00000000              dw      0000h,0000h     ;Scratch Area
=2884 C5289C28              dw      dirbuf,dpb0     ;Dir Buff, Parm Blo
=2888 64294529              dw      csv0,alv0       ;Check, Alloc Vecto
=288C AB280000      dpe1    dw      xlt1,0000h      ;Translate Table
=2890 00000000              dw      0000h,0000h     ;Scratch Area
=2894 C5289C28              dw      dirbuf,dpb1     ;Dir Buff, Parm Blo
=2898 93297429              dw      csv1,alv1       ;Check, Alloc Vecto
=                   ;               DISKDEF 0,1,26,6,1024,243,64,64,2
=   289C            dpb0    equ     offset $        ;Disk Parameter Blo
=289C 1A00                  dw      26              ;Sectors Per Track
=289E 03                    db      3               ;Block Shift
=289F 07                    db      7               ;Block Mask
=28A0 00                    db      0               ;Extnt Mask
=28A1 F200                  dw      242             ;Disk Size - 1
=28A3 3F00                  dw      63              ;Directory Max
=28A5 C0                    db      192             ;Alloc0
=28A6 00                    db      0               ;Alloc1
=28A7 1000                  dw      16              ;Check Size
=28A9 0200                  dw      2               ;Offset
=   28AB            xlt0    equ     offset $        ;Translate Table
=28AB 01070D13              db      1,7,13,19
=28AF 19050B11              db      25,5,11,17
=28B3 1703090F              db      23,3,9,15
=28B7 1502080E              db      21,2,8,14
=28BB 141A060C              db      20,26,6,12
=28BF 1218040A              db      18,24,4,10
=28C3 1016                  db      16,22
=   001F            als0    equ     31              ;Allocation Vector
=   0010            css0    equ     16              ;Check Vector Size
=                   ;               DISKDEF 1,0
=   289C            dpb1    equ     dpb0            ;Equivalent Paramet
=   001F            als1    equ     als0            ;Same Allocation Ve
=   0010            css1    equ     css0            ;Same Checksum Vect
=   28AB            xlt1    equ     xlt0            ;Same Translate Tab
=                   ;               ENDEF
=                   ;
=                   ;       Uninitialized Scratch Memory Follows:
=   28C5            begdat  equ     offset $        ;Start of Scratch A
```

```
=28C5           dirbuf  rs      128             ;Directory Buffer
=2945           alv0    rs      als0            ;Alloc Vector
=2964           csv0    rs      css0            ;Check Vector
=2974           alv1    rs      als1            ;Alloc Vector
=2993           csv1    rs      css1            ;Check Vector
=  29A3         enddat  equ     offset $        ;End of Scratch Are
=  00DE         datsiz  equ     offset $-begdat ;Size of Scratch Ar
=29A3 00                db      0               ;Marks End of Modul

 29A4           loc_stk rw  32  ;local stack for initialization
  29E4          stkbase equ offset $

  29E4          lastoff equ offset $
  02DF          tpa_seg equ (lastoff+0400h+15) / 16
  0521          tpa_len equu 0800h - tpa_seg
 29E4 00                db 0     ;fill last address for GENCMD

                ;*******************************************
                ;*                                         *
                ;*          Dummy Data Section             *
                ;*                                         *
                ;*******************************************
  0000                  dseg    0       ;absolute low memory
                        org     0       ;(interrupt vectors)
 0000           int0_offset     rw      1
 0002           int0_segment    rw      1
                ;       pad to system call vector
 0004                   rw      2*(bdos_int-1)

 0380           bdos_offset     rw      1
 0382           bdos_segment    rw      1
                        END
```

付録F

CBIOS

```
*****************************************************
*                                                   *
* This is the listing of the skeletal CBIOS which  *
* you can use as the basis for a customized BIOS   *
* for non-standard hardware.  The essential por-   *
* tions of the BIOS remain, with "rs" statements   *
* marking the routines to be inserted.             *
*                                                   *
*****************************************************

                ;*********************************************
                ;*                                           *
                ;* This Customized BIOS adapts CP/M-86 to    *
                ;* the following hardware configuration      *
                ;*     Processor:                            *
                ;*     Brand:                                *
                ;*     Controller:                           *
                ;*                                           *
                ;*                                           *
                ;*     Programmer:                           *
                ;*     Revisions :                           *
                ;*                                           *
                ;*********************************************

FFFF            true            equ -1
0000            false           equ not true
000D            cr              equ 0dh ;carriage return
000A            lf              equ 0ah ;line feed

                ;*********************************************
                ;*                                           *
                ;* Loader_bios is true if assembling the     *
                ;* LOADER BIOS, otherwise BIOS is for the    *
                ;* CPM.SYS file.                             *
                ;*                                           *
                ;*********************************************

0000            loader_bios     equ false
00E0            bdos_int        equ 224 ;reserved BDOS interrupt

                IF      not loader_bios
                ;---------------------------------------------
                ;|                                           |
2500            bios_code       equ 2500h
0000            ccp_offset      equ 0000h
0B06            bdos_ofst       equ 0B06h ;BDOS entry point
                ;|                                           |
                ;---------------------------------------------
```

```
          ENDIF   ;not loader_bios

          IF      loader_bios
     ;-----------------------------------------
     ;|                                        |
     bios_code        equ 1200h ;start of LDBIOS
     ccp_offset       equ 0003h ;base of CPMLOADER
     bdos_ofst        equ 0406h ;stripped BDOS entry
     ;|                                        |
     ;-----------------------------------------
          ENDIF   ;loader_bios

          cseg
          org     ccpoffset
     ccp:
          org     bios_code

     ;*****************************************
     ;*                                       *
     ;* BIOS Jump Vector for Individual Routines  *
     ;*                                       *
     ;*****************************************

2500 E93C00      jmp INIT       ;Enter from BOOT ROM or LOADER
2503 E97900      jmp WBOOT      ;Arrive here from BDOS call 0
2506 E98500      jmp CONST      ;return console keyboard status
2509 E98D00      jmp CONIN      ;return console keyboard char
250C E99A00      jmp CONOUT     ;write char to console device
250F E9A200      jmp LISTOUT    ;write character to list device
2512 E9B500      jmp PUNCH      ;write character to punch device
2515 E9BD00      jmp READER     ;return char from reader device
2518 E9F600      jmp HOME       ;move to trk 00 on cur sel drive
251B E9D900      jmp SELDSK     ;select disk for next rd/write
251E E90101      jmp SETTRK     ;set track for next rd/write
2521 E90301      jmp SETSEC     ;set sector for next rd/write
2524 E90C01      jmp SETDMA     ;set offset for user buff (DMA)
2527 E91701      jmp READ       ;read a 128 byte sector
252A E94701      jmp WRITE      ;write a 128 byte sector
252D E98F00      jmp LISTST     ;return list status
2530 E9F900      jmp SECTRAN    ;xlate logical->physical sector
2533 E90201      jmp SETDMAB    ;set seg base for buff (DMA)
2536 E90401      jmp GETSEGT    ;return offset of Mem Desc Table
2539 E9A400      jmp GETIOBF    ;return I/O map byte (IOBYTE)
253C E9A500      jmp SETIOBF    ;set I/O map byte (IOBYTE)

     ;*****************************************
     ;*                                       *
     ;* INIT Entry Point, Differs for LDBIOS and  *
     ;* BIOS, according to "Loader_Bios" value    *
     ;*                                       *
     ;*****************************************

     INIT:    ;print signon message and initialize hardwa
253F 8CC8     mov ax,cs          ;we entered with a JMPF so
```

```
2541 8ED0               mov ss,ax          ;CS: as the initial value o
2543 8ED8               mov ds,ax          ;DS:,
2545 8EC0               mov es,ax          ;and ES:
                        ;use local stack during initialization
2547 BC5928             mov sp,offset stkbase
254A FC                 cld                ;set forward direction

                        IF      not loader_bios
                ;---------------------------------------------
                ;|                                           |
                        ; This is a BIOS for the CPM.SYS file.
                        ; Setup all interrupt vectors in low
                        ; memory to address trap

254B 1E                 push ds            ;save the DS register
254C C606A72600         mov IOBYTE,0       ;clear IOBYTE
2551 B80000             mov ax,0
2554 8ED8               mov ds,ax
2556 8EC0               mov es,ax          ;set ES and DS to zero
                        ;setup interrupt 0 to address trap routine
2558 C70600008225       mov int0_offset,offset int_trap
255E 8C0E0200           mov int0_segment,CS
2562 BF0400             mov di,4
2565 BE0000             mov si,0           ;then propagate
2568 B9FE01             mov cx,510         ;trap vector to
256B F3A5               rep movs ax,ax     ;all 256 interrupts
                        ;BDOS offset to proper interrupt
256D C7068003060B       mov bdos_offset,bdos_ofst
2573 1F                 pop ds             ;restore the DS register

                ;       (additional CP/M-86 initialization)
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;not loader_bios

                        IF      loader_bios
                ;---------------------------------------------
                ;|                                           |
                        ;This is a BIOS for the LOADER
                        push ds            ;save data segment
                        mov ax,0
                        mov ds,ax          ;point to segment zero
                        ;BDOS interrupt offset
                        mov bdos_offset,bdos_ofst
                        mov bdos_segment,CS ;bdos interrupt segment
                ;       (additional LOADER initialization)
                        pop ds             ;restore data segment
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;loader_bios
574 BBB126
577 E86F00              mov bx,offset signon
57A B100                call pmsg          ;print signon message
57C E981DA              mov cl,0           ;default to dr A: on coldst
                        jmp ccp            ;jump to cold start entry o
```

```
257F E984DA        WBOOT:  jmp ccp+6           ;direct entry to CCP at com

                           IF    not loader_bios
                   ;-----------------------------------------------
                   ;|                                              |
                   int_trap:
2582 FA                    cli                 ;block interrupts
2583 8CC8                  mov ax,cs
2585 8ED8                  mov ds,ax           ;get our data segment
2587 BBD126                mov bx,offset int_trp
258A E85C00                call pmsq
258D F4                    hlt                 ;hardstop
                   ;|                                              |
                   ;-----------------------------------------------
                           ENDIF   ;not loader_bios

                   ;***********************************************
                   ;*                                             *
                   ;*    CP/M Character I/O Interface Routines    *
                   ;*                                             *
                   ;***********************************************

                   CONST:           ;console status
258E                       rs       10         ;(fill-in)
2598 C3                    ret

                   CONIN:                      ;console input
2599 E8F2FF                call CONST
259C 74FB                  jz CONIN            ;wait for RDA
259E                       rs       10         ;(fill-in)
25A8 C3                    ret

                   CONOUT:          ;console output
25A9                       rs       10         ;(fill-in)
25B3 C3                    ret                 ;then return data

                   LISTOUT:                    ;list device output
25B4                       rs       10         ;(fill-in)
25BE C3                    ret

                   LISTST:                     ;poll list status
25BF                       rs       10         ;(fill-in)
25C9 C3                    ret

                   PUNCH:           ;write punch device
25CA                       rs       10         ;(fill-in)
25D4 C3                    ret

                   READER:
25D5                       rs       10         ;(fill-in)
25DF C3                    ret

                   GETIOBF:
25E0 A0A726                mov al,IOBYTE
```

```
                          ret
25E3 C3
              SETIOBF:
                          mov IOBYTE,cl       ;set iobyte
25E4 880EA726             ret
25E8 C3                                        ;iobyte not implemented
              pmsg:
                          mov al,[BX]
25E9 8A07                 test al,al          ;get next char from message
25EB 84C0                 jz return
25ED 7421                 mov CL,AL           ;if zero return
25EF 8AC8                 call CONOUT
25F1 E8B5FF               inc BX              ;print it
25F4 43                   jmps pmsg
25F5 EBF2                                      ;next character and loop
              ;*****************************************
              ;*                                       *
              ;*        Disk Input/Output Routines     *
              ;*                                       *
              ;*****************************************

              SELDSK:             ;select disk given by register CL
 0002         ndisks      equu    2 ;number of disks (up to 16)
25F7 880EA826             mov disk,cl         ;save disk number
25FB BB0000               mov bx,0000h        ;ready for error return
25FE 80F902               cmp cl,ndisks       ;n beyond max disks?
2601 730D                 jnb return          ;return if so
2603 B500                 mov ch,0            ;double(n)
2605 8BD9                 mov bx,cx           ;bx = n
2607 B104                 mov cl,4            ;ready for *16
2609 D3E3                 shl bx,cl           ;n = n * 16
260B B9F126               mov cx,offset dpbase
260E 03D9                 add bx,cx           ;dpbase + n * 16
2610 C3       return:     ret                 ;bx = .dph

              HOME:   ;move selected disk to home position (Track
2611 C706A9260000         mov trk,0           ;set disk i/o to track zero
2617                      rs      10          ;(fill-in)
2621 C3                   ret

              SETTRK: ;set track address given by CX
2622 890EA926             mov trk,CX
2626 C3                   ret

              SETSEC: ;set sector number given by cx
2627 890EAB26             mov sect,CX
262B C3                   ret

              SECTRAN: ;translate sector CX using table at [DX]
262C 8BD9                 mov bx,cx
262E 03DA                 add bx,dx           ;add sector to tran table a
2630 8A1F                 mov bl,[bx]         ;get logical sector
2632 C3                   ret

              SETDMA: ;set DMA offset given by CX
```

```
2633 890EAD26              mov dma_adr,CX
2637 C3                    ret

                   SETDMAB: ;set DMA segment given by CX
2638 890EAF26              mov dma_seg,CX
263C C3                    ret

                   ;
                   GETSEGT: ;return address of physical memory table
263D BBE826                mov bx,offset seg_table
2640 C3                    ret

                   ;*********************************************
                   ;*                                           *
                   ;*  All disk I/O parameters are setup:       *
                   ;*     DISK      is disk number      (SELDSK) *
                   ;*     TRK       is track number     (SETTRK) *
                   ;*     SECT      is sector number    (SETSEC) *
                   ;*     DMA_ADR   is the DMA offset   (SETDMA) *
                   ;*     DMA_SEG   is the DMA segment  (SETDMAB)*
                   ;*  READ reads the selected sector to the DMA*
                   ;*  address, and WRITE writes the data from  *
                   ;*  the DMA address to the selected sector   *
                   ;*  (return 00 if successful,  01 if perm err)*
                   ;*                                           *
                   ;*********************************************

                   READ:
2641                       rs      50      ;fill-in
2673 C3                    ret

                   WRITE:
2674                       rs      50      ;(fill-in)
26A6 C3                    ret

                   ;*********************************************
                   ;*                                           *
                   ;*                Data Areas                 *
                   ;*                                           *
                   ;*********************************************
 26A7              data_offset     equ offset $

                           dseg
                           org     data_offset     ;contiguous with co
26A7 00            IOBYTE  db      0
26A8 00            disk    db      0       ;disk number
26A9 0000          trk     dw      0       ;track number
26AB 0000          sect    dw      0       ;sector number
26AD 0000          dma_adr dw      0       ;DMA offset from DS
26AF 0000          dma_seg dw      0       ;DMA Base Segment

                           IF      loader_bios
                   ;---------------------------------------------
                   ;|                                            |
                   signon  db      cr,lf,cr,lf
```

```
                              db      'CP/M-86 Version 1.0',cr,lf,0
                    ;|                                                   |
                    ;---------------------------------------------------
                              ENDIF   ;loader_bios

                              IF      not loader_bios
                    ;---------------------------------------------------
                    ;|                                                   |
                    signon    db      cr,lf,cr,lf
26B1 0D0A0D0A                 db      'System Generated 00/00/00'
26B5 53797374656D
     2047656E6572
     617465642030
     302F30302F30
     30
                              db      cr,lf,0
26CE 0D0A00
                    ;|                                                   |
                    ;---------------------------------------------------
                              ENDIF   ;not loader_bios

                    int_trp   db      cr,lf
26D1 0D0A                     db      'Interrupt Trap Halt'
26D3 496E74657272
     757074205472
     61702048616C
     74
                              db      cr,lf
26E6 0D0A

                    ;         System Memory Segment Table

                    segtable db 2    ;2 segments
26E8 02                       dw tpa_seg        ;1st seg starts after BIOS
26E9 C602                     dw tpa_len        ;and extends to 08000
26EB 3A05                     dw 2000h          ;second is 20000 -
26ED 0020                     dw 2000h          ;3FFFF (128k)
26EF 0020
                              include singles.lib ;read in disk definitio
=                   ;                 DISKS 2
=                   dpbase    equ     $                  ;Base of Disk Param
=  26F1             dpe0      dw      xlt0,0000h         ;Translate Table
=26F1 20270000                dw      0000h,0000h        ;Scratch Area
=26F5 00000000                dw      dirbuf,dpb0        ;Dir Buff, Parm Blo
=26F9 3A271127                dw      csv0,alv0          ;Check, Alloc Vecto
=26FD D927BA27      dpe1      dw      xlt1,0000h         ;Translate Table
=2701 20270000                dw      0000h,0000h        ;Scratch Area
=2705 00000000                dw      dirbuf,dpb1        ;Dir Buff, Parm Blo
=2709 3A271127                dw      csv1,alv1          ;Check, Alloc Vecto
=270D 0828E927      ;                 DISKDEF 0,1,26,6,1024,243,64,64,2
=                   dpb0      equ     offset $           ;Disk Parameter Blo
=  2711                       dw      26                 ;Sectors Per Track
=2711 1A00                    db      3                  ;Block Shift
=2713 03                      db      7                  ;Block Mask
=2714 07                      db      0                  ;Extnt Mask
=2715 00                      dw      242                ;Disk Size - 1
=2716 F200                    dw      63                 ;Directory Max
=2718 3F00                    db      192                ;Alloc0
=271A C0                      db      0                  ;Alloc1
=271B 00
```

```
=271C 1000              dw      16              ;Check Size
=271E 0200              dw      2               ;Offset
=  2720         xlt0    equ     offset $        ;Translate Table
=2720 01070D13          db      1,7,13,19
=2724 19050B11          db      25,5,11,17
=2728 1703090F          db      23,3,9,15
=272C 1502080E          db      21,2,8,14
=2730 141A060C          db      20,26,6,12
=2734 1218040A          db      18,24,4,10
=2738 1016              db      16,22
=  001F         als0    equ     31              ;Allocation Vector
=  0010         css0    equ     16              ;Check Vector Size
=               ;               DISKDEF 1,0
=  2711         dpbl    equ     dpb0            ;Equivalent Paramet
=  001F         alsl    equ     als0            ;Same Allocation Ve
=  0010         cssl    equ     css0            ;Same Checksum Vect
=  2720         xltl    equ     xlt0            ;Same Translate Tab
=               ;               ENDEF
=               ;
=               ;       Uninitialized Scratch Memory Follows:
=               ;
=  273A         begdat  equ     offset $        ;Start of Scratch A
=273A           dirbuf  rs      128             ;Directory Buffer
=27BA           alv0    rs      als0            ;Alloc Vector
=27D9           csv0    rs      css0            ;Check Vector
=27E9           alvl    rs      alsl            ;Alloc Vector
=2808           csvl    rs      cssl            ;Check Vector
=  2818         enddat  equ     offset $        ;End of Scratch Are
=  00DE         datsiz  equ     offset $-begdat ;Size of Scratch Ar
=2818 00                db      0               ;Marks End of Modul

 2819           loc_stk rw  32  ;local stack for initialization
   2859         stkbase equ offset $

   2859         lastoff equ offset $
   02C6         tpa_seg equ (lastoff+0400h+15) / 16
   053A         tpa_len equu 0800h - tpa_seg
 2859 00        db 0    ;fill last address for GENCMD

                ;*******************************************************
                ;*                                                     *
                ;*              Dummy Data Section                     *
                ;*                                                     *
                ;*******************************************************
 0000                   dseg    0       ;absolute low memory
                        org     0       ;(interrupt vectors)
 0000           int0_offset     rw      1
 0002           int0_segment    rw      1
                ;       pad to system call vector
 0004                   rw      2*(bdos_int-1)

 0380           bdos_offset     rw      1
 0382           bdos_segment    rw      1
                        END
```

CP/M-86
プログラマーズガイド

まえがき

このマニュアルは，ＣＰ／M-86^{TM}の環境下での8086アセンブラ言語のプログラミングを助けるものである．本書は読者がＣＰ／Ｍを含むCP/M-86に関する知識を持っていること，及び 次の Digital Research の出版物を読んでいることを前提に，書かれている．

・ＣＰ／M2　Documentation
・ＣＰ／M-86　System Guide

また，8086アセンブラ命令群についても知っていることが望ましい．これについては「Intel's 8086 Family User's Manual」に詳しく書かれている．

このマニュアルの第１章では，ＡＳＭ-86の操作方法および，起動時に指定できるアセンブラのオプションについて述べる．これらのオプションのうちの１つは，16進出力形式を制御するものである．ＡＳＭ-86は8086のマシン・コードをIntelとDigital Researchの両方の形式で出力できる．これらの２つの形式については付録Ａを参照されたい．

第２章ではＡＳＭ-86アセンブラ言語の各要素について述べる．ＡＳＭ-86で使える文字，定数，変数，識別名，演算子，式，及び 文について定義する．

第３章では，ＡＳＭ-86の擬似命令について述べる．これは条件付アセンブルを要求したり，複数のソース・ファイルからのアセンブル，出力リストの形式制御といった機能を実行する．

第４章では，ASM-86で使用できる8086の命令ニモニックを要約して述べる．このニモニックはIntelのアセンブラと，４つの命令を除いて全く同じである．４つの命令とは，セグメント内ショート・ジャンプ，セグメント外ジャンプ，リターン，コール命令である．これについては，付録Ｂを参照されたい．

第５章では，コード・マクロについて説明する．コード・マクロ定義，指定子と制御子，コード・マクロ擬似命令などについて説明する．これに関しては，付録Ｈにもまとめられている．

第６章ではＤＤＴ-86について述べる．これはユーザ・プログラムをＣＰ／M-86の環境のもとでテストしたりデバックしたりする機能を持っている．第６節にはＤＤＴ-86のデバッグ例も示されている．

目　次

1. 概論 …… 1
1.1 アセンブラの操作法 …… 1
1.2 オプション実行時パラメータ …… 3
1.3 ASM-86の強制終了 …… 4

2. ASM-86アセンブラ言語の要素 …… 5
2.1 ASM-86文字セット …… 5
2.2 最小単位と分離符 …… 5
2.3 区切り文字 …… 6
2.4 定数 …… 7
2.4.1 数値定数 …… 7
2.4.2 文字定数 …… 7
2.5 識別名 …… 8
2.5.1 キーワード …… 8
2.5.2 シンボルとその属性 …… 9
2.6 演算子 …… 11
2.6.1 演算子の使用例 …… 13
2.6.2 演算子の優先順位 …… 14
2.7 式 …… 15
2.8 文 …… 16

3. アセンブラ擬似命令 …… 17
3.1 概要 …… 17
3.2 セグメント開始擬似命令 …… 17
3.2.1 CSEG擬似命令 …… 17
3.2.2 DSEG擬似命令 …… 18
3.2.3 SSEG擬似命令 …… 18
3.2.4 ESEG擬似命令 …… 18
3.3 ORG擬似命令 …… 18
3.4 IF擬似命令とENDIF擬似命令 …… 19
3.5 INCLUDE擬似命令 …… 19
3.6 END擬似命令 …… 19
3.7 EQU擬似命令 …… 19
3.8 DB擬似命令 …… 20
3.9 DW擬似命令 …… 20
3.10 DD擬似命令 …… 21
3.11 RS擬似命令 …… 21
3.12 RB擬似命令 …… 21
3.13 RW擬似命令 …… 21
3.14 TITLE擬似命令 …… 22
3.15 PAGESIZE擬似命令 …… 22
3.16 PAGEWIDTH擬似命令 …… 22
3.17 EJECT擬似命令 …… 22
3.18 SIMFORM擬似命令 …… 22
3.19 NOLIST及びLIST擬似命令 …… 22

4. ASM-86命令セット …… 23
4.1 概要 …… 23
4.2 データ転送命令 …… 24
4.3 演算，論理，シフト命令 …… 26
4.4 ストリング命令 …… 30
4.5 制御移動命令 …… 31
4.6 プロセッサ制御命令 …… 34

5. コード・マクロ機能 …… 35
5.1 コード・マクロの概要 …… 35
5.2 指定子 …… 37
5.3 制限子 …… 37
5.4 幅指定 …… 38
5.5 コード・マクロ擬似命令 …… 38
5.5.1 SEGFIX …… 38
5.5.2 NOSEGFIX …… 38
5.5.3 MODRM …… 39
5.5.4 RELBとRELW …… 39
5.5.5 DB, DW, DD …… 40
5.5.6 DBIT …… 40

6. DDT-86 …… 42
6.1 DDT-86の操作 …… 42
6.1.1 DDT-86の起動 …… 42
6.1.2 DDT-86コマンドの規定 …… 42
6.1.3 20ビット・アドレスの指定 …… 43
6.1.4 DDT-86の終了 …… 44
6.1.5 DDT-86の割込み操作 …… 44
6.2 DDT-86コマンド …… 44
6.2.1 A(アセンブル)コマンド …… 44
6.2.2 D(表示)コマンド …… 44
6.2.3 E(実行用ロード)コマンド …… 45
6.2.4 F(充塡)コマンド …… 45
6.2.5 G(実行)コマンド …… 46
6.2.6 H(16進演算)コマンド …… 46
6.2.7 I(コマンド・テイル入力)コマンド …… 46
6.2.8 L(リスト)コマンド …… 47
6.2.9 M(転送)コマンド …… 47
6.2.10 R(読込み)コマンド …… 47
6.2.11 S(セット)コマンド …… 48
6.2.12 T(トレース)コマンド …… 48
6.2.13 U(トレースなし実行)コマンド …… 49
6.2.14 V(レイアウト表示)コマンド …… 49
6.2.15 W(書込み)コマンド …… 49
6.2.16 X(CPUステータスのチェック)コマンド …… 49
6.3 ディフォルト・セグメント値 …… 51
6.4 A／Lコマンドにおけるアセンブラの文法 …… 52
6.5 DDT-86の実行例 …… 53

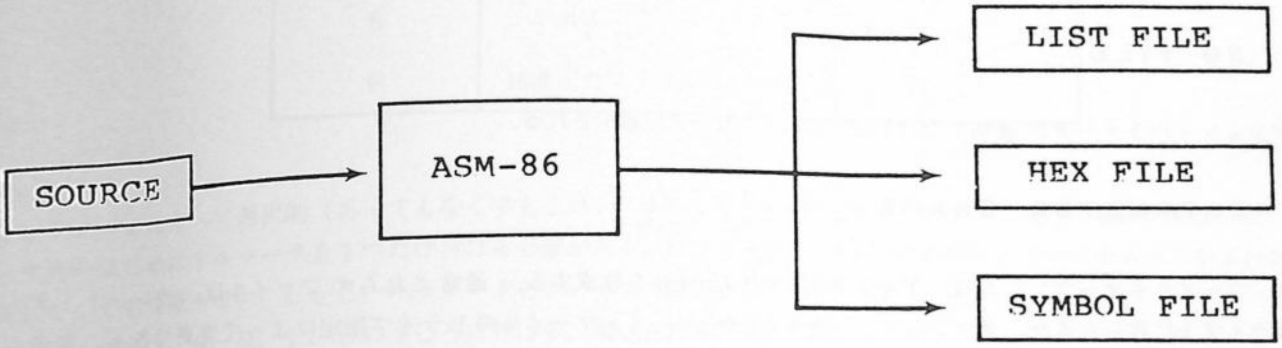
付録A. ASM-86の起動 …… 60
付録B. Intelアセンブラとの相違点 …… 62
付録C. ASM-86の16進出力形式 …… 63
付録D. 予約語 …… 65
付録E. ASM-86命令の要約 …… 66
付録F. サンプル・プログラム …… 69
付録G. コード・マクロ定義の文法 …… 74
付録H. ASM-86エラー・メッセージ …… 76
付録I. DDT-86エラー・メッセージ …… 77
ASM-86への追記 …… 78
DDT-86への追記 …… 79

第１章　概論

1.1　アセンブラの操作法

ＡＳＭ-86は，8086アセンブラ言語のソース・ファイルを３パスで処理し，16進形式の8086マシン語ファイルを含む，３つの出力ファイルを生成する．このオブジェクト・ファイルはIntel形式あるいは，Digital Resarchの16進形式のいずれでもよい．これらの形式については付録Ｃを参照のこと．ＡＳＭ-86は，8086クロスアセンブラ（インテル8080又はザイログＺ-80ベースのシステム上のＣＰ／Ｍ下で動くように設計されたもの）と，8086アセンブラ（インテル8086又は8088ベースのシステム上のＣＰ／Ｍ-86下で動くように設計されたもの）の２つの形で提供される．

ＡＳＭ-86は一般に，図１-１に示されるように１つの入力ファイルから，３つの出力ファイルを作成する．



〈ファイル名〉．A86……ソース・ファイル
〈ファイル名〉．ＬＳＴ……アセンブル・リスト
〈ファイル名〉．H86……16進形式によるオブジェクト
〈ファイル名〉．ＳＹＭ……ユーザ定義シンボルのリスト

図１-１　ＡＳＭ-86のソースとオブジェクト・ファイル

図１-１はまた，ＡＳＭ-86のファイル拡張名についてリストアップしてある．ＡＳＭ-86では任意の３文字の拡張名をつけたソース・ファイルが扱えるが，拡張名を省略して入力すると，ディレクトリの中でそのファイル名に，.A86という拡張名をつけたファイルを捜す．もしファイル名が指定されておらず，そのファイルが，.A86以外の拡張名をもっているか，あるいは拡張名を全然もっていなければ，ＡＳＭ-86はエラー・メッセージを返す．

図１-１に示すその他の拡張名は，ＡＳＭ-86の出力ファイルをあらわす．．ＬＳＴファイルはエラー・メッセージを含むアセンブル・リスト，H86はDigital Research又はIntelのどちらかの16進形式によるマシン語オブジェクトであり，．ＳＹＭはユーザ定義シンボルのリストである．

ＡＳＭ-86をスタートするには，次のコマンドを入力する．

ＡＳM86　〈ソースファイル名〉［＄〈オプションパラメータ〉］

このオプションパラメータについては１．２節で説明する．ソースファイルの指定は，次の形で指定する．

［〈ドライブ名〉：］〈ファイル名〉［．〈拡張名オプション〉］

ここで，

〈ドライブ名〉は，ソース・ファイルのある場所を示す１文字のドライブ名であり，現在のドライブにソースがあるときは不要である．

〈ファイル名〉はＣＰ／Ｍのファイル名で１から８文字．

〈拡張名オプション〉は１から３文字のファイル拡張名であり，通常は．A86である．

ＡＳＭ-86コマンドの例を挙げると，

```
A>ASM86 B:BIOS88
A>ASM86 BIOS88.ASM $FI AA HB PB SB
A>ASM86 D:TEST
```

ＡＳＭ-86がスタートすると，次のメッセージが出力される．

```
CP/M 8086 ASSEMBLER VER X.X
```

ここでX．XはＡＳＭ-86のバージョンナンバーである．ＡＳＭ-86は次に，ソース･ファイルをオープンしようとする．ファイルが指定したドライブにない場合，あるいは上に述べた正しい拡張名を持っていない場合には，次のメッセージが出力される．

```
NO FILE
```

もしオプション・パラメータが適切でなければ，次のメッセージが表示される．

```
PARAMETER ERROR
```

ソース・ファイルをオープンした後，アセンブラは出力ファイルを作成する．通常これらのファイルは，カレント・ディスク・ドライブ上に作られるが，オプション・パラメータや，ソース・ファイルのドライブ指定によって変更できる．後者の場合，ＡＳＭ-86は，出力ファイルをソース・ファイルと同じドライブ上に作成する．

アセンブル時にエラー（ディスク・フルやシンボル・テーブルのオーバフローなど）が検出されると，ＡＳＭ-86は異常終了(Abort)する．ソース・ファイルにエラーが見つかると，それはリスト・ファイル上のエラー・メッセージ行（エラーのあるソース行の直前）に書き込まれる．各エラー・メッセージは，エラー番号と簡単なエラーの説明から成り，付録Hにこのエラー・メッセージの一覧表を載せてある．アセンブルが終了すると，次のメッセージが表示される．

```
END OF ASSEMBLY. NUMBER OF ERRORS: n
```

1.2　オプション実行時パラメータ

ドルマーク「$」は，実行時パラメータ列の先頭を示す．パラメータは１文字で，うしろに同じく１文字のデバイス名が続く．次の表1-1にパラメータを示す．

表1-1　実行時パラメーター覧

パラメータ	意　味	デバイス
A	ソース・ファイル・デバイス	A，B，C……P
H	16進出力ファイル・デバイス	A…P，X，Y，Z
P	リスト・ファイル・デバイス	A…P，X，Y，Z
S	シンボル・ファイル・デバイス	A…P，X，Y，Z
F	16進出力ファイルの形式	I，D

パラメータはすべて選択的（あってもなくてもよい）であり，コマンド行の中でどのような順序で現れてもよいが，パラメータ列のはじめにドルマークを１つだけ付ける必要がある．パラメータとパラメータの間にスペースを入れてかまわないが，入れなくても支障えない．しかし，パラメータとデバイスの間には入れてはならない．

デバイス名は，A，H，P，Sのうしろに付ける．用いられるデバイス名は，

A，B，C……PとX，Y，Z

である．AからPまでは，それぞれディスク・ドライブのAからPまでを意味し，Xはユーザ・コンソール（CON：），Yはラインプリンタ（LST：）そしてZはどこにも出力しないことを意味する（NUL：）．

出力がコンソールの場合，一時的に停止させることができる．それにはコントロールSを入力すればよい．もう一度コントロールSを入力するか，どれか別のキーを打つと，再び出力を続行する．

パラメータFのうしろには，IかDのいずれかを指定しなければならない．Iを指定すると，Intel 16進形式のオブジェクト・ファイルが生成され，Dを指定すると，Digital Researchの16進オブジェクト・ファイルが生成される．これらのオブジェクトの形式については，付録Cに詳しくまとめてある．パラメータFを省略すると，Digital Research形式がとられる．

表1-2　実行時パラメータの例

コマンド行	結　　果
ASM86 IO	IO.A86というファイルがアセンブルされ，IO.HEX，IO.LST，IO.SYMがディフォルトのドライブ上に作られる．
ASM86 IO.ASM $AD SZ	ドライブD上のIO.ASMをアセンブルし，IO.LST，IO.HEXをディフォルトのドライブ上に作成する．シンボル・ファイルは作られない．
ASM86 IO $PY SX	IO.A86というファイルをアセンブルし，IO.HEXを作成し，リストをプリンタへ，シンボル・リストをコンソールへ 出力する．
ASM86 IO $FD	Digital Research 16進形式オブジェクトが作られる．
ASM86 IO $FI	Intel 16進形式オブジェクトが作られる．

1.3　ＡＳＭ86の強制終了

ＡＳＭ-86の実行を，途中で強制的に終了させるには，コンソール・キーボード上の任意のキーをたたけばよい．そうすると次の質問が出力される．

USER BREAK．OK（Y／N）？

Ｙを応答するとアセンブルは終了し，ＯＳのレベルに戻る．Ｎを入力するとそのまま処理を続行する．

第2章
ＡＳＭ-86アセンブラ言語の要素

2.1 ＡＳＭ-86文字セット

ＡＳＭ-86はいくつかの文字を除いたＡＳＣII文字を識別する．用いられるのは，英文字，数学，特殊文字，スペースといくつかのコントロール文字である．

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9

+ - * / = ( ) [ ] ; ' . ! , _ : @ $
```

スペース，タブ，キャリッジ・リターン，ライン・フィード

英文字の小文字は，文字ストリングとして使うばあいを除いて，大文字に変換して扱われる．文字ストリングとしては，英文字，数字，特殊文字とスペースが使える．（コントロール文字は使えない．）

2.2 最小単位と分離符

最小単位とは，ＡＳＭ-86のソース・プログラム上で意味をもつ最小の単位を言う．これは英文法における「単語」に相当する．最小単位と最小単位はスペースによって区切ることができる．スペースは1つでもよいし，複数個でもよい．ＡＳＭ-86では分離符として水平タブが使える．水平タブはスペースと解釈される．

タブは8カラムごとに設定されており，リスト・ファイル上で複数のスペースに置き換えられる．

2.3　区切り文字

区切り文字は，最小単位のおわりを知らせるだけの分離符に対して，命令にある意味を加味して最小単位のおわりを示す．区切り文字があると，分離符はなくてもよいが，区切り文字のあとに分離符を置いた方が，プログラムが見やすくなる．

表２－１は区切り文字と分離符の表である．区切り文字のうちのいくつかはまた，演算子でもあり，２．６節で詳しく説明する．

表2-1　分離符と区切り文字

文　字	名　　称	用　　途
20H	スペース	分離符
09H	タ　ブ	分離符．リスト・ファイル上では複数個のスペースとなる．
ＣＲ	キャリッジ・リターン(復帰)	ソース・ラインの終りを示す．
ＬＦ	ライン・フィード（改行）	ＣＲのうしろに続く．ソース行のなかではスペースとみなされる．
;	セミコロン	コメントの始まりを示す．
:	コロン	ラベル，及びセグメント越えの指定に使う．
.	ピリオド	数値から変数を形成する．
$	ドル・マーク	ロケーション・ポインタの現在の値をあらわす．
＋	プラス	加算演算子
−	マイナス	減算演算子
*	アスタリスク	乗算演算子
／	スラッシュ	除算演算子
@	アット・マーク	識別子の中で用いられる．
_	アンダーバー（下線）	識別子の中で用いられるが無視される．
!	エクスキューラメーション・マーク(感嘆符)	論理上の文の終り．これを用いて，１行に複数の文が書ける．
'	アポストロフ（引用符）	ストリング定数を示す．

2.4 定　数

定数はアセンブル時に既に与えられていて，実行時に変更されることのない値である．定数には整定数と文字ストリングがある．

2.4.1 数値定数

数値定数はある基数での16ビットの値である．基数は，数値のうしろに続く，基数インディケータによって表わす．この基数インディケータには次のようなものがある．

表2-2　基数インディケータと定数

インディケータ	定数の型	基　数
B	バイナリ	2
O	オクタル	8
Q	オクタル	8
D	デシマル	10
H	ヘキサ	16

ＡＳＭ-86はインディケータのない数値定数はすべて10進とみなす．インディケータは大文字でも小文字でもかまわない．

定数とは，１連の数字のうしろに必要ならオプションの基数インディケータがつづくものを言い，数値の部分は基数の範囲内の数字によって構成されなければならない．バイナリ定数は０か１かで，オクタル定数は０～７で，10進の場合は０～９で表わさなければならない．16進定数では，０～９の数字に加えて，A（10D），B（11D），C（12D），D（13D），E（14D），F（15D）の６文字を用いる．16進定数の数値の部分は，識別名と混同されないように０又は数字で始まる必要がある．

次にいろいろの数値定数の例を挙げておく．

```
1234     1234D    1100B    1111000011110000B
1234H    0FFEH    3377O    13772Q
33770    0FE3H    1234d    0ffffh
```

2.4.2 文字定数

引用符で囲まれたＡＳＣＩＩ文字列は文字定数として扱われる．どの命令でも，アーギュメント１～２文字の文字定数を書くことができる．その命令は，１文字を８ビットの数として扱う．２文字の場合には，はじめの１字を上位バイト，次の１字を下位バイトとして16ビットの数として扱う．

文字の値はそのＡＳＣＩＩコードの値に等しい．文字ストリングの場合にはＡＳＭ-86は小文字を大文字に変換しないので，両方の文字が使える．これらの大小英文字の他に，数字，スペースと特殊文字がストリングに使える．

ＡＳＭ-86で２文字以上の文字ストリングを扱うには，ＤＢ擬似命令を用いる．このとき，文字ストリングは255バイトを越えてはならない．文字ストリングの中で引用符自身を文字として使いたいときには，２つ続けて入力すればよい．ＡＳＭ-86は''をひとつの引用符と解釈する．

表２-３で，文字定数の例と処理後の文字ストリングの表れ方を示す．

表2-3　文字定数の例

```
            'a' -> a
        'Ab''Cd' -> Ab'Cd
   'I like CP/M' -> I like CP/M
            '''' -> '
'ONLY UPPER CASE' -> ONLY UPPER CASE
'only lower case' -> only lower case
```

2.5　識別名

識別名はアセンブラにとってある意味をもつ文字の並びである．ＡＳＭ-86の識別名はすべて，次の規則に従う．

1．最初の１文字は英字（Ａ～Ｚ，ａ～ｚ）でなければならない．
2．２文字目以降は英字でも数字（０～９）でもかまわない．特殊文字＠と＿は，使用が認められているが無視される．たとえばa__bは ab となる．
3．識別名は物理的な１行の長さを越えなければ，どんな長さでもよい．

識別名には２つの種類がある．ひとつはキーワードであり，これはすでにアセンブラに対して意味が定義されている．もうひとつはシンボルであり，これはユーザが定義する．

次に挙げるのは識別名のいろいろな例である．

```
NOLIST
WORD
AH
Third_street
How_are_you_today
variable@number@1234567890
```

2.5.1　キーワード

キーワードは既に意味が定義されている識別名である．キーワードは予約語であり，ユーザがキーワードと同じ名前の識別名を定義することはできない．キーワードの一覧表は付録Ｄを参照すること．

ＡＳＭ-86は５つのタイプのキーワードを持っている．命令，擬似命令，演算子，レジスタ，そしてアセンブラが定義している数値がそれである．このうち，8086の命令ニーモニックとその働きについては第４章で，擬似命令については第３章で，それぞれ説明する．演算子に関しては，２．６節で述べることにする．及び表2-4には，8086レジスタを表わすＡＳＭ-86キーワード のリストが挙げてある．

数値の定義されているキーワードは3つある．ＢＹＴＥ，ＷＯＲＤ，ＤＷＯＲＤで，それぞれ１，２，４の値をもつ．これらの値は変数の型属性に関係する．これらのキーワードの型属性は，キーワードの数値に等しい．型属性については２．５．２節を参照のこと．

表2-4　レジスタ・キーワード

レジスタ記号	サイズ	値	意　味
AH	1バイト	100B	アキュムレータ上位バイト
BH	〃	111B	ベース・レジスタ上位バイト
CH	〃	101B	カウント・レジスタ上位バイト
DH	〃	110B	データ・レジスタ上位バイト
AL	〃	000B	アキュムレータ下位バイト
BL	〃	011B	ベース・レジスタ下位バイト
CL	〃	001B	カウント・レジスタ下位バイト
DL	〃	010B	データ・レジスタ下位バイト
AX	2バイト	000B	アキュムレータ（1ワード）
BX	〃	011B	ベース・レジスタ（1ワード）
CX	〃	001B	カウント・レジスタ（1ワード）
DX	〃	010B	データ・レジスタ（1ワード）
BP	〃	101B	ベース・ポインタ
SP	〃	100B	スタック・ポインタ
SI	〃	110B	ソース・インデックス
DI	〃	111B	ディスティネーション・インデックス
CS	〃	01 B	コード・セグメント・レジスタ
DS	〃	11 B	データ・セグメント・レジスタ
SS	〃	10 B	スタック・セグメント・レジスタ
ES	〃	00 B	エクストラ・セグメント・レジスタ

2.5.2　シンボルとその属性

シンボルはユーザが定義する識別名である．シンボルには属性があり，それがどのような種類の情報であるかを示す．シンボルは 次の3つに分類できる．

- ◦変数
- ◦ラベル
- ◦数値

変数とは，メモリ上にあるロケーションにおかれるデータを示すもので，次の3つの属性をもつ．

- ◦セグメント――変数が定義された時にどのセグメントでアセンブルが行なわれていたかを示す．
- ◦オフセット――セグメントの始まりと，その変数のアドレスとの間のバイト数．
- ◦型――その変数が参照されたときに操作されるデータのバイト長．

セグメントにはコード・セグメント，データ・セグメント，スタック・セグメント，エクストラ・セグメントがあり，その内容とセグメントのスタート・アドレスを保持しているレジスタによって決まる（３．２節を見よ）.

セグメントは16の倍数の任意のアドレスより始まる．ＡＳＭ-86ではこの境界線の値を，変数のセグメント属性として用いる．

変数のオフセットは，０から０ＦＦＦＦＨつまり65535Ｄのあいだの値である．変数はまた，次の型属性のうちのいずれかひとつを持つ．

◦ＢＹＴＥ
◦ＷＯＲＤ
◦ＤＷＯＲＤ

ＢＹＴＥは１バイトの変数，ＷＯＲＤは２バイトの変数，ＤＷＯＲＤは４バイトの変数である．ＤＢ，ＤＷ，ＤＤ擬似命令はそれぞれ，この３つの型の変数を定義するものである（第３章参照）.

変数は例えば，ストレージを定義する擬似命令のための名前として現れたときに定義される．

```
VARIABLE DB 0
```

変数はまた，次のように別の変数を参照するＥＱＵ擬似命令のための名前としても定義される．

```
VARIABLE EQU ANOTHER_VARIABLE
```

ラベルはメモリ上の命令文のはいるロケーションを定義するもので，ジャンプ命令やコール命令で参照される．ラベルには次の２つの属性がある．

◦セグメント
◦オフセット

ラベルのセグメントとオフセットは，変数のそれと基本的には同じである．一般にはラベルは命令の前に置くことで定義される．ラベルと命令のあいだにはコロン「：」を置いて分離符とする．

```
LABEL: ADD AX,BX
```

ラベルは，別のラベルを参照するＥＱＵ擬似命令のための名前としても定義される．たとえば，

```
LABEL   EQU  ANOTHER_LABEL
```

数値もまた，シンボルとして定義できる．数値シンボルはあたかも直接数値をコーディングしたかのごとく扱われる．
例えば，

```
Number__five    EQU  5
                MOV  AL, Number__five       は，

                MOV  AL, 5         と同じである．
```

２．６節では数値と数値シンボルにかかる演算子とその効果について説明する．

2.6 演算子

ＡＳＭ-86の演算子は次のカテゴリに分けられる．算術演算子，論理演算子，比較演算子，セグメント越え指定，及び変数の生成と操作である．表２-５にＡＳＭ-86の演算子を定義してある．

表2-5　ＡＳＭ-86の演算子

文　法	結　果	制　限
	論理演算子	
a　XOR　b	aとbのビットごとの排他的論理和	a，bは数値
a　OR　b	aとbのビットごとの論理和	a，bは数値
a　AND　b	aとbのビットごとの論理積	a，bは数値
NOT　a	aの論理否定．0は1になり，1は0となる．	aは16ビットの数値
	比較演算子	
a　EQ　b	a=bなら０ＦＦＦＦＨが戻り，そうでないときは０が戻る．	a，bは符号なし数値
a　LT　b	a<bならば０ＦＦＦＦＨとなり，そうでないときは０となる．	a，bは符号なし数値
a　LE　b	a≦bならば０ＦＦＦＦＨとなり，そうでないときは０となる．	a，bは符号なし数値
a　GT　b	a>bならば０ＦＦＦＦＨとなり，そうでないときは０となる．	a，bは符号なし数値
a　GE　b	a≧bならば０ＦＦＦＦＨとなり，そうでないときは０となる．	a，bは符号なし数値
a　NE　b	a≠bならば０ＦＦＦＦＨとなり，そうでないときは０となる．	a，bは符号なし数値
	算術演算子	
a＋b	aとbとの和	aは変数，ラベル又は数値．bは数値
a－b	aとbとの差	aは変数，ラベル又は数値．bは数値．
a＊b	aとbとの符号なしの積	a，bは数値

表2-5　ASM-86の演算子

文　法	結　果	制　限
算術演算子（続き）		
a／b	aとbとの符号なしの商	a，bは数値
a　MOD　b	a/bの余りを返す．	a，bは数値
a　SHL　b	aをbビット左にシフトしたものを返す．	a，bは数値
a　SHR　b	aをbビット右にシフトしたものを返す．	a，bは数値
＋a	aを与える．	aは数値
－a	0－aを与える．	aは数値
セグメント越え		
〈セグメント・レジスタ〉：〈アドレス式〉	アセンブラの決定するセグメント・レジスタを越える．	〈セグメント・レジスタ〉はＣＳ，ＤＳ，ＳＳ，及びＥＳのいずれか．
変数操作／生成		
ＳＥＧ　a	aという変数又はラベルのセグメント値をもつ数値を生成する．	aはラベルか変数．
ＯＦＦＳＥＴ　a	aという変数又はラベルのオフセット値をもつ数値を生成する．	aはラベルか変数
ＴＹＰＥ　a	数値を生成する．変数aの型がＢＹＴＥなら1を，ＷＯＲＤなら2をＤＷＯＲＤなら4を生成する．	aはラベルか変数
ＬＥＮＧＴＨ　a	変数aのバイト長の値をもつ数値を生成する．	aはラベルか変数
ＬＡＳＴ　a	もしＬＥＮＧＴＨa>0ならＬＡＳＴaはＬＥＮＧＴＨa−1となる．ＬＥＮＧＴＨ a=0ならば，ＬＡＳＴa=0である．	aはラベルか変数
a　ＰＴＲ　b	a の型をもちb の属性をもつ仮想変数又は仮想ラベルを生成する．	aはＢＹＴＥ，ＷＯＲＤ，ＤＷＯＲＤのいずれか．bは〈アドレス式〉
．a	aというオフセットをもつ変数を生成．セグメント属性は現在のセグメントをとる	aは数値
$	ロケーション・カウンタの現在の値と等しいオフセットをもつラベルを生成する．セグメント属性は現在のセグメントとなる	アーギュメントなし

2.6.1 演算子の使用例

論理演算子はオペランドに数値しか許さない．論理演算子はブール代数論理演算AND，OR，XOR，NOTを実行する．例えば，

```
      00FC            MASK     EQU    0FCH
      0080            SIGNBIT  EQU    80H
0000  B180                     MOV    CL,MASK AND SIGNBIT
0002  B003                     MOV    AL,NOT MASK
```

比較演算子はオペランドをすべて符号のない数値として扱う．比較演算子はEQ（等しい），LT（小なり），LE（小なり又は等しい），GT（大なり），GE（大なり又は等しい），NE（等しくない）である．各々の演算子は2つのオペランドを比較して，指定された関係が「真」ならば全ビットON（0FFFFH）を，「偽」ならば0を生成する．

```
      000A            LIMIT1   EQU    10
      0019            LIMIT2   EQU    25
                               .
                               .
                               .
0004  B8FFFF                   MOV    AX,LIMIT1 LT LIMIT2
0007  B80000                   MOV    AX,LIMIT1 GT LIMIT2
```

加減演算子は2つのオペランドの和や差を計算する．最初のオペランドは変数，ラベル，数値いずれでもよいが，2番目のオペランドは数値しか許されない．数値が変数やラベルに加えられると，その結果の変数やラベルのオフセットは，第1オペランドのオフセットに第2オペランドの数値を加えたものになる．

変数／ラベルから数値を引くと，その結果の変数／ラベルのオフセットは，第1オペランドのオフセットから第2オペランドの数値をさし引いたものになる．

```
      0002            COUNT    EQU    2
      0005            DISP1    EQU    5
000A  FF              FLAG     DB     0FFH
                               .
                               .
                               .
000B  2EA00B00                 MOV    AL,FLAG+1
000F  2E8A0E0F00               MOV    CL,FLAG+DISP1
0014  B303                     MOV    BL,DISP1-COUNT
```

乗除算演算子＊，／MOD，SHL，SHRはオペランドに数値しか扱えない．＊と／はオペランドを符号なしの数値と解釈して扱う．

```
0016  BE5500                   MOV    SI,256/3
0019  B310                     MOV    BL,64/4
      0050            BUFFERSIZE      EQU      80
001B  B8A000                   MOV    AX,BUFFERSIZE * 2
```

正負オペレータは次のように符号なし，符号ありどちらのオペランドでも受付ける．

```
001E  B123                     MOV    CL,+35
0020  B007                     MOV    AL,2--5
0022  B2F4                     MOV    DL,-12
```

変数操作時には，アセンブラがどのセグメント・レジスタを用いるかを決定するが，セグメント越え演算子をべつのレジスタに指定すると，アセンブラの決定したセグメントを乗越えて操作できる．セグメント越え演算子の文法は，〈セグメント・レジスタ〉：〈アドレス式〉である．ここで〈セグメント・レジスタ〉はCS，DS，SS，ESのうちのいずれかである．例を次に示す．

```
0024 368B472D             MOV      AX,SS:WORDBUFFER[BX]
0028 268B0E5B00           MOV      CX,ES:ARRAY
```

変数操作演算子は，変数オペランドの属性のひとつに等しい数値を生成する．ＳＥＧはセグメント値を，ＯＦＦＳＥＴはオフセット値を，ＴＹＰＥは型の値（１，２，４）を，ＬＥＮＧＴＨは変数のバイト長をとりだす．ＬＡＳＴは変数のＬＥＮＧＴＨを０と比較し，０より大きければＬＥＮＧＴＨより１を引き，０に等しければ，そのまま０を返す．変数操作演算子のオペランドは変数しか許されない．

```
002D 000000000000 WORDBUFFER     DW       0,0,0
0033 0102030405   BUFFER         DB       1,2,3,4,5
                                          .
                                          .
                                          .
0038 B80500                 MOV      AX,LENGTH BUFFER
003B B80400                 MOV      AX,LAST BUFFER
003E B80100                 MOV      AX,TYPE BUFFER
0041 B80200                 MOV      AX,TYPE WORDBUFFER
```

ＰＴＲ演算子は，命令が実行されている間だけ値を保持する，仮想の変数やラベルを作り出すもので，オペランドに変化は起らない．この一時的な仮想シンボルは，次の例に示すように左側のオペランドの型属性をもち，他の属性は右側のオペランドに等しい．

```
0044 C60705                 MOV      BYTE PTR [BX], 5
0047 8A07                   MOV      AL,BYTE PTR [BX]
0049 FF04                   INC      WORD PTR [SI]
```

ピリオド演算子「．」は，現在のデータ・セグメントに変数を生成する．この新しい変数は，カレント・データ・セグメントと同じセグメント属性をもち，そのオペランドに等しいオフセット値をもつ．オペランドは数値でなければならない．

```
004B A10000                 MOV      AX, .0
004E 268B1E0040             MOV      BX, ES: .4000H
```

ドル・マーク「＄」はラベルを生成する．このラベルはロケーション・カウンタの現在の値に等しいオフセット属性をもち，そのセグメント値は現在のコード・セグメント値と同じである．オペランドは必要としない．

```
0053 E9FDFF                 JMP      $
0056 EBFE                   JMPS     $
0058 E9FD2F                 JMP      $+3000H
```

2.6.2 演算子の優先順位

式は，変数，ラベル，数値を演算子でつなぎ合せたものである．ＡＳＭ-86では，２．７節で述べるいくつかの種類の式を許されている．この節では，式の中に演算子が２つ以上現われる場合の処理の順番について説明する．

通常ＡＳＭ-86は式を左から右へと評価してゆくが，高い優先度をもつ演算子は低いものより前に評価する．２つの演算子が等しい優先度をもつときには左から評価される．

表２-６にＡＳＭ-86演算子の優先度を昇順にまとめてある．(後ほど優先度が高い)．

かっこは普通の優先度の規則を変えることができる．かっこで囲まれた式は最初に評価される．かっこが入れ子のときは，もっとも内側の式から順に処理される．かっこの入れ子は５重まで許されている．次に式の例を挙げる．

```
15/3 + 18/9 = 5 + 2 = 7
15/(3 + 18/9) = 15/(3 + 2) = 15/5 = 3
```

表2-6　ASM-86の演算優先度

順位	種　類	演　算　子
1	論理演算子	XOR, OR
2	〃	AND
3	〃	NOT
4	比較演算子	EQ, LT, LE, GT, GE, NE
5	加減算演算子	+, −
6	乗除算演算子	*, ／, MOD, SHL, SHR
7	符号演算子	+, −
8	セグメント越え	〈セグメント・レジスタ〉:
9	変数生成／操作	SEG, OFFSET, PTR, TYPE, LENGTH, LAST
10	かっこ，大かっこ	( ), 〔 〕
11	ピリオド，ドルマーク	., $

2.7 式

ASM-86では，アドレス式，数値式，大かっこ付きの式が許される．アドレス式は，メモリ・アドレスとして評価され，次の3つの情報をもつ．

- セグメント値
- オフセット値
- 型

変数も，ラベルも，アドレス式である．アドレス式は数値ではないが，その内容は数値である．数値をPTRなどの演算子と組み合せて，アドレス式に用いることもできる．

数値式は数値を評価するもので，変数もラベルも含まず，数値と演算子だけが用いられる．

大かっこの含まれる式は，ベース・アドレシング・モードか，インデックス・アドレッシング・モードかを指定するものである．ベース・レジスタはBXかBP，インデックス・レジスタはDIかSIである．大かっこの式では，ベース・レジスタ，インデックス・レジスタ，あるいは両方を用いることができる．

＋演算子をベース・レジスタとインデックス・レジスタの間に置くと，両方のアドレシングが指定できる．

```
MOV  variable[bx],0
MOV  AX,[BX+DI]
MOV  AX,[SI]
```

2.8 文

当アセンブラ言語における「最小単位」は英語の「単語」に相当するが，同様に文（ステートメント）は英語の文章（センテンス）に対応する．文はＡＳＭ-86に対して，どのような処理を行えばよいのかを指令する．文には２つの種類がある．命令と擬似命令である．命令はアセンブラによって，8086のマシン語命令に翻訳される．擬似命令は，マシン語には翻訳されないが，アセンブラに対してある機能を行うように指示を与える．

ひとつのアセンブラの文は，キャリッジ・リターン(ＣＲ)とライン・フィード（ＬＦ)，或いはエクスキュラメーション(!)によって終る．「!」は，コメント中にあるときを除いて，１つの行の終りとみなされる．「!」マークを分離符として，物理的な１行に複数の文を置くことができるわけである．

ＡＳＭ-86の命令セットについては第４章で述べる．命令文の一般的な文法は，

〔ラベル：〕〔プレフィックス〕ニーモニック〔オペランド〕〔；コメント〕

である．ここで各フィールドは次のような意味を持つ．

ラベル：

コロンで終るシンボルはラベルであり，現セグメント中のロケーション・カウンタの現在値を示す．このフィールドは指定しなくてもよい．

プレフィックス

ＬＯＣＫ，ＲＥＰのような一部の命令は，別の命令の前に書くことができる．このフィールドも，あってもなくてもよい．

ニーモニック

アセンブラに対して命令として定義されているキーワード．あるいはＥＱＵ擬似命令によって命令と定義されているシンボル．このフィールドはプレフィックスが前にない限り，あってもなくてもよい．省略されているときにはオペランドを置いてはならない．ＡＳＭ-86のニーモニックについては第４章を見よ．

オペランド

命令ニーモニックは，オペランドに別のシンボルを要求するものがある．命令は０～２個のオペランドをもつ．

コメント

文字ストリングの範囲外にセミコロンがあると，そのうしろはコメントとなり，コメントはＣＲで終る．コメントはプログラムの可読性をよくする．このフィールドはなくてもよい．

ＡＳＭ-86擬似命令については，第３章で説明する．擬似命令の一般文法は，

〔ネーム〕 擬似命令　オペランド〔；コメント〕

であり各項目は次のような意味をもつ．

ネーム

命令文のラベルとちがい，擬似命令のラベルはコロンで終ってはならない．ネームを持つことのできる擬似命令はＤＢ，ＤＷ，ＤＤ，ＲＳ，ＥＱＵである．ＤＢ，ＤＷ，ＤＤ，ＲＳについては名称はなくてもよいが，ＥＱＵについては必須である．

擬似命令

第３章でのべる擬似命令キーワードのひとつ．

オペランド

命令ニーモニックに対するオペランドと同様である．ＤＢ，ＤＷ，ＤＤ等の擬似命令は，他と違ってオペランドには制限はない．

コメント

命令文に同じ．

第3章
アセンブラ擬似命令

3.1 概　要

擬似命令文はＡＳＭ-86に内部的な機能を実行させるものである．例えば，論理的なセグメントの割り付けをしたり，条件つきアセンブリを要求したり，データ項目を定義したり，リスト・ファイルの形式を指示したりする．擬似命令文の一般形式については，２．８節で述べた．

この節では各擬似命令文がどのような文法で用いられるのかを説明する．ここでは，文法をあらわす行に，特殊記号を用いてアーギュメントに何が使えるか，或いはその他の二者択一項目を示している．大かっこ（[,]）はそのアーギュメントがオプションであることを示し，かくかっこ（<，>）はユーザが指定するアーギュメントを示す．これらの記号は，実際のコーディング時には書いてはならない．

3.2 セグメント開始擬似命令

実行時には，8086のメモリ参照命令はすべて，16ビットのセグメント・ベース値と16ビットのオフセット値をもっている必要がある．これらが組み合わさって，20ビットの実効アドレスが生成される．この実効アドレスはＣＰＵによって物理的なアドレスを決定するのに使われる．16ビットのセグメント・ベース値，即ちセグメント境界値は，セグメント・レジスタＣＳ，ＤＳ，ＳＳ，ＥＳのどれかに格納される．オフセット値は，セグメント開始アドレスからの相対アドレスである．16バイト単位のメモリ上の物理セグメントが，再配置可能な最小の単位となる．

ＡＳＭ-86では４種類の論理セグメントが定義されている．それはコード・セグメント，データ・セグメント，スタック・セグメント，そしてエクストラセグメントで，それぞれＣＳ，ＤＳ，ＳＳ，ＥＳレジスタによってアドレシングされる．

ＡＳＭ-86の将来のバージョンでは，データ・セグメントとコード・セグメントの双方に使えるいくつかの追加セグメントがサポートされる予定である．ＡＳＭ-86のすべての文は，現在ＣＰＵが参照することのできる４つのセグメントのどれかひとつに必ず割りふられる．

セグメント擬似命令ＣＳＥＧ，ＤＳＥＧ，ＳＳＥＧ，ＥＳＥＧは，それに続く文を指定セグメントに所属させるよう指示するものである．それ以降の文は，命令にセグメント越えの指定が含まれない限り，それぞれに対応したセグメント・レジスタでアドレッシングされる．
ＡＳＭ-86は別のセグメント擬似命令に出会うまでは，文を指定されたセグメントに割り当て続ける．

命令文は，コード・セグメントに割当てられる．擬似命令は，どのセグメントに割り当ててもよい．ＡＳＭ-86はこれらの命令によって，使用するセグメントレジスタを切り換える．例えば，命令がメモリ上にある変数を参照するものであるとすると，場合によってセグメント越えプレフィクス・バイトを生成する必要もあるので，ＡＳＭ-86はどのセグメントにその変数が含まれるか知っていなければならない訳である．

3.2.1 ＣＳＥＧ擬似命令

ＣＳＥＧ　〈数値式〉
ＣＳＥＧ
ＣＳＥＧ　$

この擬似命令はアセンブラに対して，以下の文はコード・セグメントに属するということを告げる．すべての命令はコード・セグメントに指定されなければならない．擬似命令の方は，コード・セグメントに所属してもしなくてもかまわない．

セグメントの位置がアセンブル時にわかっているときに，最初の形式を用いる．このときプログラムはリロケータブル（再配置可能）にならない．２番目の形式はセグメントの位置がわかっていないときに用いる．このときは，プログラムはリロケータブルになる．最後の形は他のセグメント擬似命令によってコード・セグメントがひとたび中断された後に，その続きを開始するのに用いる．継続するコード・セグメントは以前のコード・セグメントと同じ属性（ロケーション・ポインタやインストラクション・ポインタなど）を持って始まる．

3.2.2 DSEG擬似命令

DSEG 〈数値式〉
DSEG
DSEG $

この擬似命令は，以降の文がデータ・セグメントに属することを指示する．データ・セグメントは主にDB，DW，DD，RSといったデータ割り付け擬似命令を含んでいる．しかし，それ以外の擬似命令もすべて所属可能である．命令文はデータ・セグメントに入れることはできない．

最初の形式は，アセンブル時にセグメントの位置がわかっているときに用いる．この場合プログラムはリロケータブルではない．2番目の形式は，セグメントの位置がわからないときに用いる．これはリロケータブルになる．最後の形は，他のセグメント擬似命令によってデータ・セグメントが中断されたときに，その続きを開始するのに用いる．継続するデータ・セグメントは，以前のデータ・セグメントと同じ属性で始まる．

3.2.3 SSEG擬似命令

SSEG 〈数値式〉
SSEG
SSEG $

SSEG擬似命令は，スタック・セグメントのソース行のはじまりを示す．スタック・セグメントはすべてのスタック操作に使われる．すべての擬似命令はスタック・セグメントに入れることができるが，命令文は入れてはいけない．

最初の形式はアセンブル時にセグメントの位置がわかっているときに用いる．生成されたプログラムはリロケータブルでない．次の形式はセグメントの位置がわかっていないときに用いる．この場合はリロケータブルとなる．最後の形は，他のセグメント擬似命令によって中断された後に，スタック・セグメントの続きを開始するときに使う．継続するスタック・セグメントは以前のスタック・セグメントと同じ属性ではじまる．

3.2.4 ESEG擬似命令

ESEG 〈数値式〉
ESEG
ESEG $

この擬似命令はエクストラ・セグメントを初期化する．命令文はこのセグメントには置けないが擬似命令文は置くことができる． 最初の形式はアセンブル時にセグメントの位置がわかっているときに用いる．この場合，プログラムはリロケータブルにな らない．2番目の形式は，セグメントの位置がわかっていないときに用いる．このときはリロケータブルになる．最後の形は，他のセグメント擬似命令によって中断された後に，再びエクストラ・セグメントの続きを開始するときに用いる．継続する拡張セグメントは以前の拡張セグメントの属性をそのまま引継ぐ．

3.3 ORG擬似命令

ORG 〈数値式〉

ORG擬似命令は，現セグメント中のロケーション・カウンタのオフセット値を，〈数値式〉で指定した値にセットする．

ORG擬似命令よりも前に，式の要素をすべて定義しておく必要がある．というのは，この文より先にある要素を参照しようとしても，不確定だからである．

たいていのセグメントでは，ORG文は不必要である．セグメントの最初の命令又はデータ・バイト以前にORG文がない場合，アセンブルはセグメントの先頭から相対ゼロの位置より開始される．ひとつのセグメントの中にORG擬似命令はいくつ現れてもよい．

3.4 IF擬似命令とENDIF擬似命令

```
IF 〈数値式〉
    〈ソース行1〉
    〈ソース行2〉
       ・
       ・
    〈ソース行n〉
    ENDIF
```

IFとENDIF擬似命令は，ソース行の1群をアセンブルの対象からはずしたり含めたりすることを許す．この条件付の擬似命令は，ひとつのソース・プログラムからいくつかの異ったバージョンをアセンブルするのに使う．

アセンブラがIF文を見つけると，そのIF のあとに続く式を評価する．そして式の値が0でなければ〈ソース行1〉から〈ソース行n〉までをアセンブルする．式の値が0のときには，リストは行なわれるがアセンブルは行なわれない．〈数値式〉に含まれる要素はすべて時前に定義されていなければならない．IF擬似命令の入れ子は許されない．

3.5 INCLUDE擬似命令

```
INCLUDE 〈ファイル名〉
```

この擬似命令は，ソース・テキスト中にもうひとつのASM-86のソース・ファイルの内容を展開する．例えば，

```
INCLUDE EQUALS.A86
```

INCLUDE文は，ソース・プログラムがいくつかの異なるファイルに分割して入っているときに使う．INCLUDE文は入れ子には使えない．つまり，INCLUDEによってコールされるソース・ファイルはその中にINCLUDE文を含んでいてはならない．〈ファイル名〉に拡張名がないときは .A86とみなされる．〈ファイル名〉にドライブ名の指定がないと，もとのソース・ファイルのあるドライブとみなされる．

3.6 END擬似命令

```
END
```

END擬似命令は，ソース・ファイルの終りを示す．このあとにソース行があっても無視される．END文はなくてもかまわない．ないときはEND-OF-FILE(1AH) をもってソースの終わりと見なす．

3.7 EQU擬似命令

```
シンボル EQU 〈数値式〉
シンボル EQU 〈アドレス式〉
シンボル EQU 〈レジスタ〉
シンボル EQU 〈命令ニーモニック〉
```

EQU擬似命令は，ユーザの定義するシンボルに，値と属性を与える．このシンボルは，コロンをつけてはならない．シンボルは，EQUや他の擬似命令で定義し直すことはできない．数値式やアドレス式に含まれる要素は，この文よりも前に定義されているものでなければならない．

最初の形式は，シンボルに数値を割当てるもので，次の形式は，シンボルにアドレスを割当てる．3番目の形式は，8086のレジスタに，別の名前をつけるときに使う．最後の形式では，命令セットに新しい名前を付ける．次の例はこれらの4つの形式の使い方を示す．

```
0005                FIVE     EQU     2*2+1
0033                NEXT     EQU     BUFFER
0001                COUNTER  EQU     CX
                    MOVVV    EQU     MOV
                                     .
                                     .
                                     .
005D 8BC3                    MOVVV   AX,BX
```

3.8 DB擬似命令

〔シンボル〕 DB 〈数値式〉〔,〈数値式〉…〕

〔シンボル〕 DB 〈文字定数〉〔,〈文字定数〉…〕

DB擬似命令は，バイト形式の格納領域を初期化する．数値式は8ビットの値として評価され，16進出力ファイルに順番に配置される．文字定数は2.4.2節で述べた規則に従って出力ファイルに書かれる．DB擬似命令は，2バイト以上の文字定数を扱えるASM-86で唯一の擬似命令である．この文字定数の小文字は，大文字に変換されない．カンマで区切った複数の式又は定数が物理的な1行を越えない範囲で，いくつでも指定できる．

シンボルはプログラムを通して，このデータ格納領域を参照するときに用いる．このシンボルは，4つの属性をもつ．セグメント属性とオフセット属性はシンボルのメモリ上の位置を示し，型属性はBYTE型を，データ長は確保する領域のバイト数を示す．

次にシンボルつきのDB擬似命令の例をあげる．

```
005F 43502F4D2073 TEXT   DB      'CP/M system',0
     797374656D00
006B E1           AA     DB      'a' + 80H
006C 0102030405   X      DB      1,2,3,4,5
                                 .
                                 .
                                 .
0071 B90C00              MOV     CX,LENGTH TEXT
```

3.9 DW擬似命令

〔シンボル〕 DW 〈数値式〉〔,〈数値式〉…〕

〔シンボル〕 DW 〈文字定数〉〔,〈文字定数〉…〕

DW擬似命令は2バイト（1ワード）単位の格納領域を初期化する．文字定数は，2桁までで，それ以上は使えない．それ以外はDWはDBと同じく領域の初期化を行うが，最初に下位バイトに格納され，次に上位バイトに格納される点が異なる．

次に示すのはDW文を使った例である．

```
0074 0000         CNTR   DW      0
0076 63C166C169C1 JMPTAB DW      SUBR1,SUBR2,SUBR3
007C 010002000300        DW      1,2,3,4,5,6
     040005000600
```

3.10 ＤＤ擬似命令

〔シンボル〕 ＤＤ 〈数値式〉〔，〈数値式〉…〕

ＤＤ擬似命令は，４バイト単位の格納領域を初期化する．アドレス式のオフセット属性が下位の２バイトに格納され，セグメント属性が上位の２バイトに格納される．その他は，ＤＢと同じである．

```
1234                             CSEG   1234H
                                  .
                                  .
                                  .
0000 6CC134126FC1 LONG_JMPTAB    DD     ROUT1,ROUT2
     3412
0008 72C1341275C1                DD     ROUT3,ROUT4
     3412
```

3.11 ＲＳ擬似命令

〔シンボル〕 ＲＳ 〈数値式〉

ＲＳ擬似命令はメモリ上にデータ領域を確保するが，初期化は行わない．数値式が確保するバイト数を与える．ＲＳ文はそのシンボルにバイト属性を与えない．

```
0010          BUF    RS    80
0060                 RS    4000H
4060                 RS    1
```

3.12 ＲＢ擬似命令

〔シンボル〕 ＲＢ 〈数値式〉

ＲＢ擬似命令はメモリ上にバイト単位の領域確保するが，初期化は行わない．この擬似命令はバイト属性を与えるという点を除いて，ＲＳ擬似命令と同じである．

3.13 ＲＷ擬似命令

〔シンボル〕 ＲＷ 〈数値式〉

ＲＷ擬似命令はメモリ上に２バイト（１ワード）単位の格納領域を確保し，初期化は行わない．数値式は確保するワード数となる．

```
4061          BUFF   RW    128
4161                 RW    4000H
C161                 RW    1
```

3.14 TITLE擬似命令

TITLE 〈文字定数〉

ASM-86は，TITLE文によって定義された文字定数をリスト・ファイルの各ページの頭に出力する．タイトルの文字ストリングは30文字を越えてはならない．

```
TITLE  'CP/M monitor'
```

3.15 PAGESIZE擬似命令

PAGESIZE 〈数値式〉

PAGESIZE文はリストファイル出力を1ページあたり何行にするかを定義するもので，デフォルト値は66である．

3.16 PAGEWIDTH擬似命令

PAGEWIDTH 〈数値式〉

PAGEWIDTH文は，リスト・ファイルが出力されるときの1行に入るカラム数を定義する．デフォルト値は120であるが，コンソールに直接出力するときには，デフォルトは79となる．

3.17 EJECT擬似命令

EJECT

EJECT擬似命令があると，印字中にその箇所で改頁を実行する．EJECT文そのものは，次の頁の頭に印字される．

3.18 SIMFORM擬似命令

SIMFORM擬似命令は，印字ファイル中のフォーム・フィード（FF）文字を，正確な数のライン・フィード（LF）に置き換える．この擬似命令はプリンタにフォーム・フィード機能が付いていないときに使う．

3.19 NOLIST及びLIST擬似命令

NOLIST
LIST

NOLIST擬似命令があると，以降の行は出力されない．次に現れたLIST文から，出力が再開される．

第4章
ＡＳＭ-86命令セット

4.1 概　要

ＡＳＭ-86の命令セットは，すべての8086マシン語命令を含んでいる．２.７節で命令文の一般形式を示したので，ここではラベルの参照とコメントを除く各命令について，オペランドのタイプと文法を詳しく説明する．

各命令の定義は，１覧表にまとめてある．さらに詳しい説明はインテル社の，「ＭＣＳ-86 Assembly Language Reference Manual」を参照すること．命令のビット・パターンと操作については，インテル社の，「MCS-86 User's Manual」を参照されたい

命令定義表は，ＡＳＭ-86命令文がニーモニックとオペランドの組み合せの形で書かれている．ニーモニックとは，ひとつの命令を覚え易い形に記号化したもので，オペランドとは，その命令が必要とするパラメータである．
命令によってオペランドのないもの，オペランドが１つのもの，オペランドが２つのものがある．オペランドが２つの場合，左のオペランドが命令操作の受け側となる．２つのオペランドはカンマで区切る．

命令定義表は，ＡＳＭ-86の命令セットを機能別に分けて書かれており，各表の内部ではニーモニックのアルファベット順に並べてある．
命令定義表のオペランド・タイプに用いてある記号の意味を，表4-1に示す．

表4-1　オペランド・タイプ記号

記　号	オペランド・タイプ
numb	すべての数値式
numb 8	８ビットに評価される数値式
acc	アキュムレータ．ＡＸ又はＡＬ
reg	汎用レジスタ．セグメント・レジスタを除く．
reg16	16ビットの汎用レジスタ．セグメント・レジスタを除く．
segreg	セグメント・レジスタ．ＣＳ，ＤＳ，ＳＳ，ＥＳのいずれか．

mem	アドレス式．ベース・アドレッシング・モードかインデックス・アドレッシング・モード，あるいはその双方が含まれてもよい．
	`variable`
	`variable+3`
	`variable[bx]`
	`variable[SI]`
	`variable[BX+SI]`
	`[BX]`
	`[BP+DI]`

simpmem	アドレス式．ベース・アドレシング・モード，インデックス・アドレッシング・モードを含まない．
	`variable`
	`variable+4`

mem|reg	「mem」でも「reg」でもよい．
mem|reg16	「mem」でも「reg」でもよい．ただし，16bits.
label	ラベルを表わすアドレス式．
lab 8	「label」に同じ．但し現在のアドレスから±128バイトの範囲内に限られる．

8086のＣＰＵは１ビットのフラグ・レジスタを９個持っており，それぞれがＣＰＵの状態を反映する．ユーザは直接このフラグ・レジスタをアクセスできないがそれを見て，実行された命令がレジスタやオペランドに及ぼした効果について知ることができる．命令がフラグ・レジスタに与える効果は，表4-2に示す9つのフラグ・レジスタ記号を用いて，命令定義表に述べてある．

表4-2　フラグ・レジスタ記号

ＡＦ	補助キャリ・フラグ
ＣＦ	キャリ・フラグ
ＤＦ	ディレクション・フラグ
ＩＦ	インタラプト・イネーブル・フラグ
ＯＦ	オーバフロー・フラグ
ＰＦ	パリティ・フラグ
ＳＦ	符号フラグ
ＴＦ	トラップ・フラグ
ＺＦ	ゼロ・フラグ

4.2　データ転送命令

データ転送命令には，汎用，アキュムレータ，アドレス対象，フラグの４つの種類がある．フラグを書き替えられるのはＳＡＨＦとＰＯＰＦだけである．表４-３では，acc＝ＡＬのときには１バイトしか転送されず，acc＝ＡＸのときには１ワード転送されるので注意されたい．

表４-３　データ転送命令

文　法	実 行 結 果
IN acc, numb 8 | numb16	numb 8 又はnumb16（０～255）で与えられる入力ポートより，データをアキュムレータに送る．
IN acc, DX	ＤＸレジスタ（０～０ＦＦＦＦＨ）によって与えられる入力ポートから，データをアキュムレータに送る．
LAHF	ＳＦ，ＺＦ，ＡＦ，ＰＦ，ＣＦフラグをＡＨレジスタへ転送する．
LDS reg16, mem	メモリ・アドレス（ＤＷＯＲＤ変数）のセグメント値の部分をＤＳレジスタに送り，オフセット値の部分を16ビット汎用レジスタに送る．
LEA reg16, mem	メモリ・アドレスのオフセットを，16ビットのレジスタへ転送する．
LES reg16, mem	メモリ・アドレスのセグメント値をＥＳレジスタへ，オフセット値を16ビット汎用レジスタへ転送する．

表4-3（続き）

文法	実行結果
MOV reg, mem|reg	メモリ又はレジスタを，レジスタへ転送する．
MOV mem|reg, reg	レジスタを，メモリ又はレジスタへ転送する．
MOV mem|reg, numb	直接数値データを，メモリ又はレジスタへ転送する．
MOV segreg, mem|reg16	メモリ又はレジスタを，セグメント・レジスタへ転送する．
MOV mem|reg16, segreg	セグメント・レジスタを，メモリ又はレジスタへ転送する．
OUT numb 8 | numb16, acc	numb 8 又はnumb16（0～255）で与えられている出力ポートへ，アキュムレータからデータを転送する．
OUT DX, acc	ＤＸレジスタで与えられている出力ポートへ，アキュムレータからデータを転送する．
POP mem|reg16	スタックの１番上の要素をメモリ又はレジスタへ転送する．
POP segreg	スタックの１番上の要素をセグメント・レジスタへ転送する． ただし，ＣＳ（コード・セグメント）レジスタへの転送は許されていない．
POPF	スタックの１番上の要素をフラグ・レジスタへ転送する．
PUSH mem|reg16	メモリ又はレジスタを，スタックの１番上にプッシュする．
PUSH segreg	セグメント・レジスタの内容をスタックへプッシュする．
PUSHF	フラグをスタックへプッシュする．
SAHF	ＡＨレジスタの内容をフラグへ転送する．
XCHG reg, mem|reg	レジスタと，メモリ又はレジスタの内容を，入れ替える．
XCHG mem|reg, reg	メモリ又はレジスタと，レジスタの内容を，入れ替える．
XLAT mem|reg	メモリ又はレジスタで与えられるテーブルのサーチを開始する．レジスタ場合は必ずＢＸである．ＡＬはＢＸからのオフセットＡＬにおき替わる．

4.3 演算，論理，シフト命令

8086は四則演算をいろいろな方法で実行する．演算は，８ビットでも16ビットでもよいし，符号なしでも符号付でもよい．

９個の１ビット・フラグのうちの６個は，たいていの算術演算によって（その演算の結果いかんに従って）セット／クリアされる．

表４－４に，演算命令のフラグへ及す効果をまとめた．表４－５は演算命令の，表４－６は論理命令とシフト命令の定義表である．

表４－４　フラグへ及ぼす効果

ＣＦ	演算結果が，上位ビットにけた上り（加算時）又は上位ビットよりけた下り（減算時）するときにセットされる．そうでないときは，クリアされる．
ＡＦ	演算結果が，下位４ビットにけた上り（加算時）又はけた下り（減算時）するときにセットされる．そうでないときは，クリアされる．
ＺＦ	演算結果がゼロのときにセットされ，ゼロでないときにはクリアされる．
ＳＦ	演算結果が負のときにセットされる．
ＰＦ	演算結果の下位の８ビットが，偶数パリティならばセットされ，奇数パリティならばクリアされる．
ＯＦ	演算結果がオーバフロー（結果のサイズが受け手側のサイズより大きい）のときにセットされる．

表４－５　演算命令

文　法	実　行　結　果
AAA	加算を非圧縮ＢＣＤ（ASCII）に調整する．ＡＬが調整される．
AAD	除算を非圧縮ＢＣＤ（ASCII）に調整する．ＡＬが調整される．
AAM	乗算を非圧縮ＢＣＤ（ASCII）に調整する．ＡＸが調整される．
AAS	減算を非圧縮ＢＣＤ（ASCII）に調整する．ＡＬが調整される．
ADC reg，reg|mem	メモリ又はレジスタの内容をレジスタに加える（キャリ有り）．
ADC mem|reg，reg	レジスタの内容を，メモリ又はレジスタに加える（キャリ有り）．
ADC mem|reg，numb	直接数値をメモリ又はレジスタに加える（キャリ有り）．
ADD reg，mem|reg	メモリ又はレジスタの内容をレジスタに加える．
ADD mem|reg，reg	レジスタの内容をメモリ又はレジスタに加える．

表4-5（続き）

文法	実行結果
ADD mem|reg, numb	直接数値をメモリ又はレジスタに加える.
CBW	ＡＬバイトを符号拡張してＡＸ１ワードに変換する.
CWD	ＡＸ１ワードを符号拡張して，ＡＸ／ＤＸ．２ワードに変換する.
CMP reg, mem|reg	メモリ又はレジスタと，レジスタを比較する.
CMP mem|reg, reg	レジスタと，メモリ又はレジスタを比較する.
CMP mem|reg, numb	定数とメモリ又はレジスタを比較する.
DAA	加算を10進に調整する．ＡＬが調整される.
DAS	減算を10進に調整する．ＡＬが調整される.
DEC mem|reg	メモリ又はレジスタから１を引く.
INC mem|reg	メモリ又はレジスタに１を足す.
DIV mem|reg	アキュムレータＡＸ又はＡＬを，メモリ又はレジスタ（符号なし）で割る．商がバイトのときはＡＬに商，ＡＨに余りが入り，ワードのときはＡＸに商，ＤＸに余りが入る.
IDIV mem|reg	アキュムレータＡＸ又はＡＬを，メモリ又はレジスタ（符号つき）で割る．商と余りはDIVに同じ.
IMUL mem|reg	メモリ又はレジスタにアキュムレータＡＸ又はＡＬを(符号付き)でかける．積はバイト演算のときＡＨ，ＡＬに入り，ワード演算のときＤＸ，ＡＸに入る.
MUL mem|reg	メモリ又はレジスタにアキュムレータＡＸ又はＡＬを(符号なし)でかける．積はIMULに同じ.
NEG mem|reg	メモリ又はレジスタの２の補数をとる.
SBB reg, mem|reg	メモリ又はレジスタの内容をレジスタから引く（けた下り有り）.
SBB mem|reg, reg	レジスタの内容を，メモリ又はレジスタから引く（けた下り有り）.
SBB mem|reg, numb	直接数値をメモリ又はレジスタから引く（けた下り有り）.
SUB reg, mem|reg	メモリ又はレジスタの内容をレジスタから引く.
SUB mem|reg, reg	レジスタの内容を，メモリ又はレジスタから引く.
SUB mem|reg, numb	直接数値をメモリ又はレジスタから引く.

表4-6　論理命令とシフト命令

文　法	実　行　結　果
AND reg, mem|reg	レジスタと，メモリ又はレジスタのビット毎の論理積をとる．
AND mem|reg, reg	メモリ又はレジスタとレジスタとのビット毎の論理積をとる．
AND mem reg, numb	メモリ又はレジスタと，定数との論理積をとる．
NOT mem|reg	メモリ又はレジスタの，１の補数をとる．
OR reg, mem|reg	レジスタと，メモリ又はレジスタの，ビット毎の論理和をとる．
OR mem|reg, reg	メモリ又はレジスタと，レジスタの，ビット毎の論理和をとる．
OR mem|reg, numb	メモリ又はレジスタと定数との，ビット毎の論理和をとる．
RCL mem|reg, 1	レジスタ又はメモリを，キャリ・フラグを通して１ビット左に回転する．
RCL mem|reg, CL	メモリ又はレジスタを，キャリ・フラグを通して，ＣＬレジスタで与えられるビット数だけ左に回転する．
RCR mem|reg, 1	メモリ又はレジスタを，キャリ・フラグを通して，１ビット右に回転する．
RCR mem|reg, CL	メモリ又はレジスタを，キャリ・フラグを通して，ＣＬレジスタで与えられるビット数だけ右に回転する．
ROL mem|reg, 1	メモリ又はレジスタを１ビット左に回転する．
ROL mem|reg, CL	メモリ又はレジスタをＣＬレジスタで与えられるビット数だけ左に回転する．
ROR mem|reg, 1	メモリ又はレジスタを１ビット右に回転する．
ROR mem|reg, CL	メモリ又はレジスタをＣＬレジスタで与えられるビット数だけ右に回転する．
SAL mem|reg, 1	メモリ又はレジスタを１ビット左へシフトする．下位ビットにはゼロが入る．
SAL mem|reg, CL	メモリ又はレジスタをＣＬレジスタで与えられるビット数だけ左にシフトする．下位ビットにはゼロが入る．
SAR mem|reg, 1	メモリ又はレジスタを１ビット右へシフトする．上位ビットにはもとの最上位ビットの値が入る．

表4-6（続き）

文法	実行結果
SAR mem|reg, CL	メモリ又はレジスタをCLレジスタで与えられるビット数だけ右にシフトする．上位ビットにはもとの最上位ビットの値が詰まる．
SHL mem|reg, 1	メモリ又はレジスタを1ビット左へシフトする．下位ビットにはゼロが入る．ＳＨＬとＳＡＬは別のニーモニックである．
SHL mem|reg, CL	メモリ又はレジスタをCLレジスタで与えられるビット数だけ左へシフトする．下位ビットにはゼロが入る．SHLとSALは別のニーモニックである．
SHR mem|reg, 1	メモリ又はレジスタを1ビット右へシフトする．上位ビットにはゼロが入る．
SHR mem|reg, CL	メモリ又はレジスタをCLで与えられるビット数だけ右へシフトする．上位ビットにはゼロが入る．
TEST reg, mem|reg	レジスタと，メモリ又はレジスタとの論理積をとる．結果のいかんによってフラグはセットされるが，受け側のレジスタは変化しない．
TEST mem|reg, reg	レジスタ又はメモリと，レジスタとの論理積をとる．結果のいかんによってフラグはセットされるが，受け側のメモリ又はレジスタは変化しない．
TEST mem|reg, numb	定数とメモリ又はレジスタの論理積をとる．結果のいかんによってフラグはセットされるが，メモリ又はレジスタは変化しない．
XOR reg, mem|reg	レジスタと，メモリ又はレジスタの排他的論理和をとる．
XOR mem|reg, reg	メモリ又はレジスタと，レジスタとの排他的論理和をとる．
XOR mem|reg, numb	メモリ又はレジスタと定数との排他的論理和をとる．

4.4　ストリング命令

ストリング命令は1つないし2つのオペランドを持つ．オペランドはそのオペランド･タイプで定義してあるが，そのタイプによって操作が1バイトづつ行われたり，1ワードづつ行われたりする．オペランドが2つある場合，ソース･オペランド（右）はSIレジスタでアドレシングされ，受け側オペランド（左）はDIレジスタでアドレシングされる．アドレシングは必ずＳＩ，ＤＩレジスタで行なわれる．ストリング操作の時には，ＤＩでアドレシングされる受け側オペランドは必ずエクストラ・セグメント（ＥＳ）上になければならないことに注意．

表4-7　ストリング命令

文　法	実　行　結　果
CMPS mem|reg, mem|reg	受け側からソース・オペランドを引く．フラグへ影響するだけで結果は返さない．
LODS mem|reg	1バイト又は1ワードをソース・オペランドからアキュムレータへ送る．
MOVS mem|reg, mem|reg	1バイト又は1ワードをソースから受け側へ転送する．
SCAS mem|reg	アキュムレータ（ＡＸ又はＡＬ）からオペランドを引く．フラグへ影響するが結果は返さない．
STOS mem|reg	アキュムレータからオペランドへ1バイト又は1ワードを送る．

表4-8に，ストリング命令のプレフィクスを定義した．プレフィクスがあると，ＣＸレジスタの与える回数だけ，ストリング命令をくり返す．1回くり返されるたびに，ＣＸの値からは1が引かれる．プレフィクスのニーモニックは，2.8節で示したように，ストリング命令のニーモニックの直前に置く．

表4-8　プレフィクス命令

文　法	結　　果
REP	ＣＸレジスタがゼロになるまでくり返す．
REPZ	ＣＸレジスタがゼロになり，かつゼロ・フラグ（ＺＦ）がゼロでなくなるまでくり返す．
REPE	ＲＥＰＺと同じ．
PEPNZ	ＣＸレジスタがゼロになり，かつゼロ・フラグ（ＺＦ）がゼロになるまでくり返す．
REPNE	ＲＥＰＮＺと同じ．

4.5 制御移動命令

制御移動命令には次の4種類がある.

- ○コール，ジャンプ，リターン
- ○条件付ジャンプ
- ○くり返し制御
- ○割込み

制御移動命令は，プログラムの実行をメモリ上のある新しい位置（別のコード・セグメントも可）へ渡す．制御の移動には，絶対的に移動するものと，ある決まった条件に従って行われるものとがある．表4-9に，これらの命令を定義する．条件付ジャンプのところで，「上である」とか「下である」とかは，符号なしの値の間の関係を表わし，「大きい」とか「小さい」とかは，符号付きの値の間の関係を表わす．

表4-9　制御移動命令

文　法	実　行　結　果
CALL label	次の命令のオフセット・アドレスをスタックへプッシュし，ラベルで示されるアドレスへジャンプする．
CALL mem/reg16	次の命令のオフセット・アドレスをスタックへプッシュし，メモリ又はレジスタの内容によって示されるアドレスへジャンプする．
CALLF label	セグメント・レジスタCSをスタックへプッシュし，次の命令のオフセット・アドレスをそのあとにプッシュして，ラベルで示されるアドレスにジャンプする．
CALLF mem	セグメント・レジスタCSをスタックへプッシュし，次の命令のオフセット・アドレスをそのあとにプッシュして，2ワードのメモリ領域の内容で示されるアドレスへジャンプする．
INT numb8	フラグ・レジスタをスタックへプッシュし（PUSHFと同じ），TFとIFをクリアし，256個の割込みベクトルのひとつを用いて，間接修飾により制御を移動する．スタックを3レベル使用する．
INTO	OF（オーバフロー・フラグ）がセットされているとき，フラグ・レジスタをスタックへプッシュし（PUSHFと同じ）TFとIFをクリアし，割込みベクタの4番目の要素（10H）を用いて，間接修飾により制御を移動する．OFがセットされていなければ，何も起こらない．
IRET	以前に割込みによってセーブされたアドレスへリターンする．CSとIPと一緒に，フラグレジスタも以前にストアされていたものを再ロードする．スタックは3レベル戻る．
JA lab8	「等しいか下でない」とき又は「上である」とき（（CForZF）=0）ジャンプする．

表4-9（続き）

文　法	実　行　結　果
JAE lab 8	「下でない」又は「等しいか上である」とき（ＣＦ＝０）ジャンプする.
JB lab 8	「下である」又は「等しいか上でない」とき（ＣＦ＝１）ジャンプする.
JBE lab 8	「等しいか下である」又は「上でない」とき（（ＣＦorＺＦ）＝１）ジャンプする.
JC lab 8	JBに同じ.
JCXZ lab 8	ＣＸレジスタがゼロならばラベルにジャンプする.
JE lab 8	「等しい」又は「ゼロである」とき(ＺＦ＝１）ジャンプする.
JG lab 8	「等しいか小さくない」又は「大きい」とき（（（ＳＦ xorＯＦ）orＺＦ）＝０）ジャンプする.
JGE lab 8	「小さくない」あるいは「等しいか大きい」とき（（ＳＦ xor ＯＦ）＝０）ジャンプする.
JL lab 8	「小さい」あるいは「等しいか大きくない」とき（（ＳＦ xor ＯＦ)＝１）ジャンプする.
JLE lab 8	「等しいか小さい」あるいは「大きくない」とき（（（ＳＦ xor ＯＦ）orＺＦ）＝１）ジャンプする.
JMP label	ラベルへジャンプする.
JMP mem|reg16	メモリ又はレジスタの内容が示すアドレスへジャンプする.
JMPF label	ラベルへジャンプする．ラベルは別のコード・セグメントでもよい.
JMPS lab 8	現在のアドレスから±128バイト以内にあるラベルへジャンプする.
JNA lab 8	JBEに同じ.
JNAE lab 8	JBに同じ.
JNB lab 8	JAEに同じ.
JNBE lab 8	JAに同じ.
JNC lab 8	JNBに同じ.
JNE lab 8	「等しくない」あるいは「ゼロでない」とき（ＺＦ＝０）ジャンプする.
JNG lab 8	JLEに同じ.
JNGE lab 8	JLに同じ.

表4-9（続き）

文法	実行結果
JNL lab8	JGEに同じ．
JNLE lab8	JGに同じ．
JNO lab8	「オーバフローでない」（OF＝0）ときジャンプする．
JNP lab8	「パリティなし」あるいは「奇数パリティ」のときジャンプする．
JNS lab8	「符号なし」のときジャンプする．
JNZ lab8	JNEに同じ．
JO lab8	「オーバフロー」のとき（OF＝1）ジャンプする．
JP lab8	「パリティ」あるいは「偶数パリティ」のとき（PF＝1）ジャンプする．
JPE lab8	JPに同じ．
JPO lab8	JNPに同じ．
JS lab8	「符号あり」のとき（SF＝1）ジャンプする．
JE lab8	JEに同じ．
LOOP lab8	CXレジスタより1を引き，ゼロにならなければジャンプする．
LOOPE lab8	CXレジスタより1を引き，ゼロにならず，ZFフラグがセットされていればジャンプする．「ゼロでなくなるまでループ」あるいは「等しくなくなるまで ループ」．
LOOPNE lab8	CXレジスタから1を引き，ゼロにならず，ZFフラグがクリアされていればジャンプする．「ゼロになるまでループ」あるいは「等しくなるまでループ」．
LOOPNE lab8	LOOPNEに同じ．
LOOPZ lab8	LOOPEに同じ．
RET	以前にCALL命令によってプッシュされたリターン・アドレスへ返る．スタック・ポインタは2増加する．
RET numb	以前にCALL命令によってプッシュされたリターン・アドレスへ戻る．スタック・ポインタは2＋numb増加する．
RETF	以前にCALLF命令によってプッシュされたリターン・アドレスへ戻る．スタック・ポインタは4増加する．
RETF numb	以前にCALLF命令によってプッシュされたリターン・アドレスへ戻る．スタック・ポインタは4＋numb増加する．

4.6 プロセッサ制御命令

プロセッサ制御命令は，フラグ・レジスタを操作する．さらに，これらの命令のうちのいくつかは，8086ＣＰＵと外部装置との同期をとることができる．

表4-10 プロセッサ制御命令

文 法	実 行 結 果
CLC	ＣＦフラグのクリア
CLD	ＤＦフラグをクリアし，ストリング命令のオペランドのポインタを自動増加させる．
CLI	IFフラグをクリアして，外部マスカブル 割込みを不可にする．
CMC	ＣＦフラグを反転する．
ESC numb 8，mem|reg	実効アドレスを計算し，それをアドレス・バスに配置する．(ＥＳＣは8087演算用］・プロセッサで使われる）その他には何もしない．numb 8は０～63の値でなくてはならない．
LOCK	プレフィクス命令である．8086プロセッサは,この後ろに続く命令の実行中「バス・ロック」信号を出し続ける．ＬＯＣＫプレフィクスはどの命令の前に置いてもよい．バス・ロックはコ・プロセッサがバスを使用するのを妨げる．これは，資源を分割して使用するのに有効である．
HLT	8086プロセッサを，割込みがかかるまで停止状態にする．
STC	ＣＦフラグをセットする．
STD	ＤＦフラグをセットして，ストリング命令のオペランド・ポインタを自動減小する．
STI	IFフラグをセットし，外部マスカブル 割込みを可能にする．
WAIT	「テスト」端子に信号が来ていなければ，8086プロセッサを「ウエイト」状態にする．

第5章
コード・マクロ機能

5.1 コード・マクロの概要

ＡＳＭ-86は伝統的なアセンブラ言語のマクロをサポートしていないが，ユーザに独自の命令を定義することを許している．それには，コード・マクロ擬似命令を用いる．

通常のマクロのように，コード・マクロはアセンブラのプログラム中どこにでも，現れた時点でアセンブルされるが，類似性はそこで終る．通常のマクロはアセンブラ言語をいくつか含むが，コード・マクロはただコード・マクロ擬似命令だけを含む．マクロは通常，ユーザのシンボル・テーブルに登録されるが，ＡＳＭ-86のコード・マクロはアセンブラのシンボル・テーブルに登録される．マクロは，同一の命令の列をプログラム中で何回も使うときに，それを簡略化するものである．しかし，コード・マクロはビット列を出力ファイルへ送り，新しい命令をアセンブラに追加するのである．

ＡＳＭ-86はコード・マクロを命令としてとり扱うので，ユーザはコード・マクロを，プログラム中に命令として置くことができる．次の例は，コード・マクロで定義済みの命令ＭＡＣを使ったものである．

```
        .
        .
        .
XCHG BX,WORD3
MAC  PAR1,PAR2
MUL  AX,WORD4
        .
        .
        .
```

ここでＭＡＣは２つのオペランドを持っている．ＭＡＣが定義された時に，この２つのオペランドの型，サイズ，その他の属性は，ＭＡＣの形式パラメタの定義によって決められている．形式パラメータの名前は固定されてはいない．それらはコード・マクロが使われた時にオペランドとして与えられる名前や値で，置き替えられる仮の名前である．つまり形式パラメータとは，「場所を確保」し，オペランドがどこでどのように使われる予定なのかを示すのである．

コード・マクロの定義はコード・マクロ名とその形式パラメータの書かれた行で始まる．

Code Macro 〈名前〉〔〈形式パラメータの列〉〕

そして〈形式パラメータの例〉の内容は，

〈形式名〉：〈指定子〉〔〈制限子〉〕〔〈幅〉〕

このとき〈形式名〉は固定の名前ではなく，ただ場所を確保するためのものである．形式パラメータの列が指定されるときには，指定子は必須であり制限子はオプションである．

指定子にはＡ，Ｃ，Ｄ，Ｅ，Ｍ，Ｒ，Ｓ，Ｘが，制限子にはｂ，ｄ，ｗ，sbが用いられる．アセンブラは，文字列として用いる以外は，大文字か小文字かの違いを無視するが，わかりやすくするためにこの章では指定子を大文字，制限子を小文字で示すことにする．次の節以降で，これらの指定子，制限子，そして幅については説明する．

コード・マクロの本体の部分では，ビット・パタンと形式パラメータの記述を行う．次の擬似命令がコード・マクロ中で許されている．

```
SEGFIX
NOSEGFIX
MODRM
RELB
RELW
DB
DW
DD
DBIT
```

これらの擬似命令は，コード・マクロ内では特別の意味を持ち，他の部分に同じ名前のＡＳＭ-86擬似命令（ＤＢ，ＤＷ，ＤＤ）があってもそれはコード・マクロ中でのものとは別の意味を持つ．これらのコード・マクロ擬似命令については，後の節で説明する．コード・マクロの終りは次の様な行で示す．

```
EndM
```

CodeMacro，EndM，そしてコード・マクロ擬似命令はすべて予約語である．コード・マクロ定義の文法は，付録Ｇにバッカス記法でまとめてある．次の例はコード・マクロ定義の代表的なものである．

```
CodeMacro AAA
  DB 37H
EndM

CodeMacro DIV divisor:Eb
  SEGFIX divisor
  DB     6FH
  MODRM  divisor
EndM

CodeMacro ESC opcode:Db(0,63),src:Eb
  SEGFIX src
  DBIT 5(1BH),3(opcode(3))
  MODRM  opcode,src
EndM
```

5.2　指定子

形式パラメータには必ず指定子を置かねばならない．これは，その形式パラメータの位置に，どのようなタイプのオペランドが要求されているのかを示す．表5－1はとり得る8つの指定子をまとめたものである．

表5－1　コード・マクロのオペランド指定子

文字	オペランドの型
A	アキュムレータ．AL又はAX．
C	コード．ラベル表現に限られる．
D	直接数値データ．
E	実効アドレス．メモリ（M）又はレジスタ（R）．
M	メモリ・アドレス．変数でも大かっこ付のレジスタ式でもよい．
R	汎用レジスタのみ．
S	セグメント・レジスタのみ．
X	直接メモリ参照．

5.3　制限子

オペランドの制限子は，オペランドにさらに要求を加えるものである．制限子の意味は，オペランドのタイプによって変わる．変数の場合，制限子は，オペランドに次のような型を要求する．「b」はバイト，「w」はワード，「d」は2ワード，「sb」は符号付バイトである．数値の場合，制限子は要求する数値のサイズを意味する．「b」は－256～255の数，「w」はそれ以外の数である．

表5－2にコード・マクロ制限子をまとめてある．

表5－2　コード・マクロのオペランド制限子

変数		数値	
制限子	タイプ	制限子	サイズ
b	バイト	b	－256　～255
w	ワード	w	その他
d	2ワード		
sb	符号付バイト		

5.4　幅指定

オプションの「幅」は，かっこの中に１つ又は２つの式をカンマで区切って指定される．　次のような組合せがありうる．

```
(numberb)
(register)
(numberb,numberb)
(numberb,register)
(register,numberb)
(register,register)
```

Numberbは８ビットの数値で，アドレスではない．次の例では，入力ポートをＤＸレジスタで指示するように定義している．

```
CodeMacro IN dst:Aw,port:Rw(DX)
```

次の例は，ローテーションのカウンタにＣＬレジスタを用いるように定義している．

```
CodeMacro ROR dst:Ew,count:Rb(CL)
```

最後の例では，「opcode」は直接数値で指定され，０～63の範囲内であることを定義している．

```
CodeMacro ESC opcode:Db(0,63),adds:Eb
```

5.5　コード・マクロ擬似命令

コード・マクロ擬似命令は，ビット・パターンを定義し，オペランドがどのように扱われるのかを指定する．これらの擬似命令は予約語であり，アセンブラの命令と同じものもいくつかあるが，コード・マクロ定義の中では別の意味をもっている．コード・マクロ擬似命令としては，以下の９種類のみが許されている．

5.5.1　ＳＥＧＦＩＸ

ＳＥＧＦＩＸが有ると，与えられたメモリ・アドレスをアクセスするのに，セグメント越えのプレフィクス・バイトが必要かどうかが調べられる．必要ならば，それは命令の最初のバイトとして出力される．必要なければ何も起らない．ＳＥＧＦＩＸのとる形式は，次のようなものである．

ＳＥＧＦＩＸ 〈形式名〉

ここで〈形式名〉は，メモリ・アドレスをあらわす形式パラメタの名前である．メモリ・アドレスをあらわすのであるから，その形式パラメータには，Ｅ，Ｍ，Ｘのいずれかが指定されなければならない．

5.5.2　ＮＯＳＥＧＦＩＸ

ＮＯＳＥＧＦＩＸは，オペランドのセグメントにＥＳレジスタを用いる必要のある命令のために使う．これはＣＭＰＳ，ＭＯＶＳ，ＳＣＡＳ，ＳＴＯＳ命令の受け側オペランドだけに適用される．ＮＯＳＥＧＦＩＸの形式は，

ＮＯＳＥＧＦＩＸ　segreg，〈形式名〉

である．segregはセグメント・レジスタＥＳ，ＣＳ，ＳＳ，ＤＳのいずれかであり，〈形式名〉はメモリ・アドレスを表す形式パラメータの名前である．この形式パラメータはむろん，ＥかＭかＸが指定されている必要がある．この擬似命令からは，コードは発生しないが，エラー・チェックは行なわれる．次のコーディングは，ＮＯＳＥＧＦＩＸを使った例である．

```
CodeMacro MOVS si_ptr:Ew,di_ptr:Ew
  NOSEGFIX       ES,di_ptr
  SEGFIX         si_ptr
  DB             0A5H
EndM
```

5.5.3 MODRM

この擬似命令はアセンブラに対し，ModRMバイトを生成するように命ずる．このバイトの後ろにはいろいろな8086の命令コードのバイトが続く．ModRMバイトとは，インデックス・タイプ或いは命令中で使われるレジスタ・ナンバーその他の情報を含むものである．

ModRMバイトは３つのフィールドの情報を持っている．modフィールドは，バイトのはじめの２ビットを占め，レジスタ／メモリ・フィールドと組み合わさって，８つのレジスタと24のインデックス・モード，合計32種類の値を形成する．

レジスタ・フィールド（reg）は，medフィールドのうしろの３ビットを占める．これはレジスタ・ナンバー，又は命令コードの追加３ビットの情報である．レジスタ・フィールドの意味は，命令コード・バイトによって決まる．

レジスタ／メモリ・フィールドは，このバイトの最後の３ビットを占める．このフィールドには，オペランドのロケーションを示すレジスタが入るか，或いは上に述べたmodフィールドとの組み合わせで，アドレス・モードの一部を形成する．

8086の命令とそのビット・パターンについてさらに詳しい情報は，インテル社の「8086 Assembly Language Programing Manual」と，「Intel 8086 Family User's Manual」を参照されたい．ＭＯＤＲＭの形式は次の通り．

ＭＯＤＲＭ　〈形式名〉，〈形式名〉
ＭＯＤＲＭ　ＮＵＭＢＥＲ７，〈形式名〉

ＮＵＭＢＥＲ７は０～７の数値であり，〈形式名〉は形式パラメタの名前である．次の例はＭＯＤＲＭの使い方を示す．

```
CodeMacro RCR dst:Ew,count:Rb(CL)
  SEGFIX         dst
  DB             0D3H
  MODRM          3,dst
EndM

CodeMacro OR dst:Rw,src:Ew
  SEGFIX         src
  DB             0BH
  MODRM          dst,src
EndM
```

5.5.4 ＲＥＬＢとＲＥＬＷ

これらの擬似命令は，ＩＰ相対分岐命令で使われるが，命令の終りとオペランドであるラベルとの間の変位を生成する．ＲＥＬＢで生成される変位（相対アドレス）は１バイトであり，ＲＥＬＷで生成される変位は１ワードである．形式は次の通り．

ＲＥＬＢ〈形式名〉
ＲＥＬＷ〈形式名〉

ここで〈形式名〉はＣ（コード）指定子による形式パラメータの名前である．使用例を次に挙げる．

```
CodeMacro LOOP place:Cb
  DB             0E2H
  RELB           place
EndM
```

5.5.5 DB, DW, DD

これらの擬似命令は，コード・マクロの外部で使うときとは異なる意味をもつ．形式は次のようになる．

DB〈形式名〉| NUMBERB
DW〈形式名〉| NUMBERW
DD〈形式名〉

ここで，NUMBERBは1バイトの数値，NUMBERWは2バイトの数値，〈形式名〉は形式パラメータの名前である．次に例を挙げる．

```
CodeMacro XOR dst:Ew,src:Db
  SEGFIX      dst
  DB          81H
  MODRM       6,dst
  DW          src
EndM
```

5.5.6 DBIT

この擬似命令は，1バイトかそれ以下のビットの組合せを操作するもので，形式は，

DBIT〈フィールド記述〉〔,〈フィールド記述〉〕

である．この〈フィールド記述〉には次の2つの形がある．

〈数値〉〈組合せ〉
〈数値〉(〈形式名〉(〈右シフト〉))

ここで〈数値〉はセットするビット数であり，1から16までの値である．〈組合せ〉は望みのビットの組合せを示す．フィールド記述の中の〈数値〉の合計は，16を越えてはならない．上に示した2番目の形は〈形式名〉を含んでいるが，これは指定した位置にある決った値を置く命令の形式パラメーターの名前である．その数は通常，コード・マクロの1行目で指定したレジスタを参照するものである．ここではその数は次のように各レジスタに対応している．

```
AL:   0
CL:   1
DL:   2
BL:   3
AH:   4
CH:   5
DH:   6
BH:   7
AX:   0
CX:   1
DX:   2
BX:   3
SP:   4
BP:   5
SI:   6
DI:   7
ES:   0
CS:   1
SS:   2
DS:   3
```

一番内側のかっこの中の〈右シフト〉は，シフトするビット数を示す．0が指定されていればシフトせず，1が指定されていると右へ1ビットシフトする，等々．

次の例はこの形を使っている．

```
CodeMacro DEC dst:Rw
  DBIT 5(9H),3(dst(0))
EndM
```

バイトのはじめの5ビットの値は9Hである．もしうしろのビットがゼロならば，そのバイトの16進値は48Hである．命令，

```
DEC DX
```

がアセンブルされ，DXが値2Hをもつとすると，48H＋2H＝4AHとなり，これがこのバイトの実行時の値である．もしこの手順が次のような定義をされていたら，

```
DBIT 5(9H),3(dst(1))
```

そのときはレジスタナンバーは右へ1ビットシフトされ，48H＋1H＝49Hとなってエラーとなる．

第6章
DDT-86

6.1 DDT-86の操作

DDT-86™プログラムは，CP／M-86の下でユーザ・プログラムのテストとデバッグを会話的にできるようにしたものである．これを使用するには，8086プロセッサ，ASM-86，CP／M-86のOSについて知識が必要である．CP／M-86のOSについては，「CP／M-86 System Guide」に詳しい．

6.1.1 DDT-86の起動

DDT-86は，次のコマンドのうちの1つを入力すると起動される．

DDT86
DDT86　ファイル名

上のコマンドは，DDT-86を単にロードして実行するだけである．開始メッセージが表示され，プロンプト（>）が出力されて，操作コマンドが入力可能となる．

下のコマンドは，上のコマンドと同じであるが，DDT-86がロードされた後，ファイル名で指定されたファイルがロードされる．ファイル名にファイル拡張名が省略されているときは，.CMDが補われる．DDT-86では.H86ファイルをロードすることはできない．下の形のコマンドは，次の手順を行うことと等しい．

A>DDT86
DDT86　X.X
－Eファイル名

この時点でロードされたプログラムが実行可能となる．

6.1.2 DDT-86コマンドの規定

DDT-86がコマンド入力可能のときには，ハイフン（－）が表示されて入力を促す．それに応じて，コマンド行又はコントロールC（この章では↑Cと書く）を入力する．↑Cを入力した時には，DDT-86を終了する（6.1.4節参照）．コマンド行は64文字以内でなければならず，CR(キャリッジ・リターン)で終る．コマンドをキーボードから入力している間は，CP／Mの標準ライン・エディタ機能（↑X，↑H，↑Rなど）が入力ミスの修正に使える．DDT-86はCRを入力するまでは，コマンド行の処理を開始しない．

各コマンド行の最初の1字がコマンドの動作を決定する．表6-1はDDT-86コマンドの要約である．個々のコマンドについては，6.2節で定義する．

表6-1　DDT-86コマンドの要約

コマンド	動　作
A	アセンブラ文の入力.
D	メモリを16進とASCIIで表示する.
E	実行のためのプログラム・ロード.
F	メモリのある領域に定数を詰める.
G	実行をスタートする. ブレーク・ポイントも設定可.
H	16進演算.
I	ファイル・コントロール・ブロックとコマンド・テイルの設定.
L	8086ニーモニックを使ってメモリをリスト・アップする.
M	メモリ上のある領域を転送する.
R	ディスク・ファイルをメモリに読み込む.
S	メモリに値をセットする.
T	プログラムの実行をトレースする.
U	プログラムを実行するがトレースしない.
V	ディスクから読込んだプログラムのメモリ・レイアウトを表示する.
W	メモリのある領域の内容をディスクに書込む.
X	CPUのステータスをチェック/修正する.

コマンドの後ろには1個ないしそれ以上のアーギュメントが続く.それは16進値であったり，ファイル名であったり，その他の情報であったり，コマンドによって様々である. アーギュメントはカンマかスペースで区切るが，コマンドと第1アーギュメントの間には，スペースを入れてはならない.

6.1.3　20ビット・アドレスの指定

DDT-86コマンドのほとんどが，オペランドに1個以上のアドレスをもつ. 8086は1メガバイトのメモリまでアドレシングできるので，20ビットのアドレス値が必要となる.

20ビット・アドレスは次のように入力する.

SSSS：OOOO

SSSSは16ビットのセグメント値であり，必要な場合とそうでない場合がある. OOOOは16ビットのオフセット値である. DDT-86はこの2つの値を組み合せて，次のように20ビットの実効アドレスを作り出す.

```
     SSSS0
  +   OOOO
  ---------
     eeeee
```

オプションのＳＳＳＳは，16ビットの16進数又はセグメント・レジスタの名前でもよい．セグメント・レジスタ名を使ったときには，ＳＳＳＳの値はＣＰＵのユーザ・ステータスでの該当レジスタの値となる．Xコマンドで表示されるときも同様である．

ＳＳＳＳを省略すると，実行されているコマンドに対応するデフォルト値が使われる．これについては6.4節で述べる．

6.1.4 ＤＤＴ-86の終了

ＤＤＴ-86は，ハイフンが表示されているときに↑Ｃを入力すると終了する．終了すると制御はＣＣＰに戻る．ＣＰ／Ｍ-86は，（８ビット・マシンのＣＰ／Ｍにあるような）ＳＡＶＥ機能を持たない．ファイルのパッチにＤＤＴ-86を使うときには，ＤＤＴ-86を終了する前に，Ｗコマンドを用いてディスクに書き戻さなければならない．

6.1.5 ＤＤＴ-86での割込み操作

ＤＤＴ-86は，割込みを可能にしたり不可にしたりする．ＤＤＴ-86下で実行を開始すると，開始時のプログラムの割込み状態は保存される．ＤＤＴ-86に制御が戻ったときは，当初のＤＤＴ-86起動時と，割込みフラグの状態は同じである．テスト対象のプログラムがＣＰＵの制御を持っている間は，そのＣＰＵステータスが割込みフラグのステータスを決定する．ただし，緊急の部分的割り込みができない場合を除いてである．

6.2 ＤＤＴ-86コマンド

この節ではＤＤＴ-86コマンドとそのアーギュメントを詳しく説明する．ＤＤＴ-86コマンドは，プログラムの実行を制御したり，メモリやＣＰＵのステータスを表示／修正したりすることができる．

6.2.1 Ａ（アセンブル）コマンド

Ａコマンドは8086のニーモニックを，直接アセンブルしてメモリへ配置する．形式は，

As

である．ここで，ｓはアセンブルの開始アドレスを示す，20ビット・アドレスである．Ａコマンドを入力すると，メモリ上のアセンブル開始アドレスが表示される．この時点で，４章で述べた文法に従って，アセンブラの文を入力する．文が入力し終わるとＤＤＴ-86はそれをマシンコードに変換し，その結果をメモリに配置し，次の配置可能なメモリ・アドレスを表示する．このプロセスは，ピリオドを入力するか，ＣＲのみを返すかするまで続く．

ＤＤＴ-86は，間違った文を入力すると，クエスチョン・マーク（？）を表示して，再度アドレスを表示し直す．

6.2.2 Ｄ（表示）コマンド

Ｄコマンドは，メモリの内容を８ビット又は16ビットの16進表現，及びASCIIで表示する．形式は次のいずれかである．

D
Ds
Ds，f
DW
DWs
DWs，f

ここでｓは，表示を開始する20ビット・アドレスであり，ｆはｓのセグメントの中の，16ビットの終了オフセットである．メモリは１行ないし数行にまたがって表示される．各行には，メモリ・ロケーションの16バイト分かそれ以下が表示される．

最初の3つの形式では，表示される1行は次の形になる．

ssss:oooo bb bb . . . bb cc . . . c

ssssは開始セグメント，ooooはオフセットである．bbは各バイトの16進値であり，cは各バイトの，対応するASCII文字である．印字できないASCII文字は，ピリオドに置き換えられる．

最初の形式に対しては，DDT-86は現表示アドレスから12行を表示する．2番目の形式に対しては，まず20ビット表示アドレスをsにセットしてから，12行を表示する．3番目の形式では，sからfまでの内容が表示される．うしろの3つの形式は，前の3つと同じであるが，ただメモリの内容が8ビットの替わりに16ビットの値で書かれる点が異なる．

ssss:oooo wwww wwww . . . wwww cccc . . . cc

表示が長いときには，コンソールの任意のキーをたたけば，Dコマンドは途中で終了する．

6.2.3 E（実行用ロード）コマンド

Eコマンドはファイルをメモリへロードして，後で説明するG，T，Uコマンドがプログラムの実行を開始できるようにする．Eコマンドは次の形式をとる．

E<ファイル名>

ファイル名はロードするファイルの名前であり，拡張名のないときには，CMDが補われる．セグメント・レジスタとIPレジスタの内容は，ロードされたファイルのヘッダー情報に従って修正される．

Eコマンドは，DDT-86下で実行されていたプログラムによって，あるいは以前にEコマンドやRコマンドによって確保されていたメモリ領域を解放する．従って，一度ロードし実行できるファイルはひとつだけである．

ロードが完了するとロードされたファイルの各セグメントのスタートとエンド・アドレスが表示される．この情報はあとでVコマンドによっても得られる．

ファイルが存在しないときとか，有効メモリ内にロードしきれないときには，エラー・メッセージが返される．

6.2.4 F（充塡）コマンド

Fコマンドは，メモリのある領域に，バイト単位又はワード単位に定数を詰め込む．形式は次の通り．

Fs, f, b
FWs, f, w

sは詰め込む領域の20ビット・スタート・アドレスであり，fは終りのバイトを示し，Sで指定されたセグメントにある16ビットのオフセット・アドレスである．

第1の形に対しては，8ビットの値bがsからfまで格納される．第2の形に対しては，16ビットの値wが，sからfまで格納される．順番は，最初が下位8ビット，次に上位8ビットの順である．

sがfより大きいか，bが255より大きいときには，クエスチョン・マークが返される．不可能なRAMのアドレスを指定したり，格納した値が正しく再読込みされなかったときには，エラー・メッセージが出力される．

6.2.5 G（実行）コマンド

Gコマンドはテストするプログラムに制御を渡し，必要な場合には1箇所ないし2箇所のブレーク・ポイントを設定することができる．形式は，

```
G
G,b1
G,b1,b2
Gs
Gs,b1
Gs,b1,b2
```

である．sは実行を開始する20ビット・アドレスで，b1とb2はブレーク・ポイントの20ビット・アドレスである．これらの3つのアドレスにセグメント値が省略されているときには，デフォルトはCSレジスタの内容となる．

初めの3つの形式では，スタート・アドレスが指定されていないが，CSレジスタとIPレジスタから20ビット・アドレスが計算される．

第1の形は，ブレーク・ポイントなしで実行する．第2，第3の形ではそれぞれ，1箇所又は2箇所のブレーク・ポイントを設定してから，実行を開始する．

後の3つの形は，最初の3つとほぼ同じで，まずsの値をCSとIPにセットしてから実行に入る．

ひとたび制御がテストするプログラムに渡ると，ブレーク・ポイントに出会うまで実時間で実行される．ブレーク・ポイントに出会うと，再びDDT-86の制御に戻り，全てのブレーク・ポイントがクリアされて，ブレークのかかった現アドレスが表示される．

```
*ssss:oooo
```

このときssssとooooはそれぞれ，ブレークが起こった時のCSとIPの値である．DDT-86に制御が戻った時点では，このアドレスの命令はまだ実行されていない．

6.2.6 H（16進演算）コマンド

Hコマンドは，2つの16ビット値の和と差を計算する．

```
Ha,b
```

aとbは計算の対象となる16進値である．和（ssss）と差（dddd）は次のように表示される（16ビットを越えた部分は捨てられる）．

```
ssss dddd
```

6.2.7 I（コマンド・テイル入力）コマンド

Iコマンドは，DDT-86のベース・ページにあるファイル・コントロール・ブロックとコマンド・テイル・バッファを設定し，Eコマンドで最後にロードされていたファイルのベースページにその情報をコピーする．形式は次の通り．

```
I<コマンド・テイル>
```

<コマンド・テイル>は通常1つないしそれ以上のファイル名から成る文字ストリングである．最初のファイル名は，005CHにある標準ファイル・コントロール・ブロックに対応する．2番目のファイル名は（もしあれば）006CHから始まる2番目の標準ファイル・コントロール・ブロックに対応する．<コマンド・テイル>中の文字は，0080Hにある標準コマンド・バッファにコピーされる．0080Hに<コマンド・テイル>のバイト長が入り，そのうしろに文字ストリングが続き，最後はバイナリ・ゼロで終る．

Eコマンドによって既にファイルがロードされていると，ファイル・コントロール・ブロックとコマンド・バッファは，ロードされていたプログラムのベース・ページへもコピーされる．DDT-86のベース・ページの位置は，DDT-86起動時のSSレジスタから得られる．Eコマンドでロードされていたプログラムのベース・ページは，ロード終了時のDSの値である．

6.2.8 L（リスト）コマンド

Lコマンドはアセンブラ言語でメモリの内容を出力する．形式は，

L

Ls

Ls,f

である．sは出力を開始する20ビット・アドレス，fはsで指定されたセグメント内にある16ビット・オフセットであり，出力はそこで終る．第1の形を入力すると，現在のリスト・アドレスから，マシン・コードを逆アセンブルしたものが12行出力される．第2の形を入力すると，まずリスト・アドレスをsにセットしてから，同じく12行出力する．最後の形では，sからfまで逆アセンブルしたものを出力する．どの場合にも，リスト・アドレスは，出力終了後自動的にまだ出力されていない次の行へ移動する．制御が・テスト中のプログラムからDDT-86に戻るときには（G，T，Uコマンドなど），リスト・アドレスは現在のCSとIPの値にセットされる．

出力が長いときは，途中で任意のキーをたたくことによって，Lコマンドを終了できる．あるいは↑Sを入力すると，表示出力が1時的に停止する．

Lコマンドで出力されるアセンブラ言語の文法は，第4章に従う．

6.2.9 M（転送）コマンド

Mコマンドは，メモリ上のあるデータ領域を別の領域へ転送する．形式は，

Ms,f,d

である．sは転送する領域のスタート・アドレス（20ビット），fは転送する領域のsで指定されたセグメントにある（転送を終了する）最後のバイトのオフセット（16ビット）である．dは，データを転送される側のスタート・アドレス（20ビット）である．dでセグメントを省略すると，sと同じセグメントがとられる．もしdが，sとfのあいだにあると，転送はsから順に行われるので，転送の途中で領域の1部が書き替えられてしまうことに注意．

6.2.10 R（読込み）コマンド

Rコマンドは，ファイルをメモリの連続した領域に読込む．形式は，

R<ファイル名>

である．ファイル名は読込むファイルの名前と拡張名である．

DDT-86は，ファイルをメモリに読込み，そのファイルによって占められるメモリ上の領域のスタート／エンド・アドレスを表示する．この情報はのちに，Vコマンドによって得ることもできる．表示ポインタ（Dコマンドで意味をもつ）はこのとき，読み込まれたファイルの領域の先頭にセットされる．

Rコマンドはすでに RコマンドやEコマンドで読込まれたメモリ領域を解放しない．つまり，メモリ上に重なることなく複数のファイルが読込めることになる．読込めるファイルの数は7個以内である．これは，BDOSで割り合てられる数から，DDT-86自身の分を1引いた数である．

ファイルが存在しなかったり，ロードするのに十分なメモリが残っていなかった場合には，エラー・メッセージが出力される．

6.2.11 S（セット）コマンド

Sコマンドは，ワード単位やバイト単位にメモリにある値をセットする．形式は，

Ss
SWs

である．sはセットされる20ビット・アドレスである．これを入力すると，次の行にアドレスと現内容が表示される．最初の形式に対しては，

ssss：oooo　bb

と表示し，2番目の形に対しては，

ssss：oooo　wwww

と出力する．bbは1バイト分の，wwwwは1ワード分のメモリ内容である．

上のような出力に対して，ユーザは，メモリの書き替えをするか，そのままにしておくかを選択することができる．16進の値を入力すればその値に書き替えられ，何も入力せずCRすると，メモリは変化せずに，次のアドレスとその内容を出力してくる．どちらの場合もピリオドが入力されるか，不適切な値が入力されるまで，アドレスと内容を表示し続ける．

DDT-86はメモリに格納した値が正しく再読込みできなかったり，不可能なRAMのアドレスを指示したりすると，エラー・メッセージを出力する．

6.2.12 T（トレース）コマンド

Tコマンドは，プログラムの実行を1～0FFFFHステップ単位でトレースする．形式は，

T
Tn
TS
TSn

である．nは実行する命令の数である．DDT-86は命令を実行する前に，現在のCPUステータスと，逆アセンブルした命令を表示する．最初の2つの形を入力した場合，セグメント・レジスタは表示されない．CPUステータスは1行に収まる．うしろの2つの形を使うと，1行目に全レジスタの状態が表示され，2行目に逆アセンブルされた命令が表示される．表示の形式は，Xコマンドによる出力と同じである．

どの形式でも，実行はCS，IPで示されるアドレスより開始される．nが指定されていないときには，1命令だけ実行される．指定されているときは，n命令が実行され，各ステップの実行前にCPUステータスが表示される．トレースが長いときには，任意のキーをたたいて，出力を途中で終了させることができる．

Tコマンドが終了したときには，リスト・アドレス（Lコマンドで用いる）は，まだ実行されていない次のアドレスにセットされる．

DDT-86はBDOSの割込み命令を通さずにトレースを行う．DDT-86自身がBDOSをコールしており，BDOSはリエントラントではないからである．そのかわり，BDOSからリターンする割込みの手続きは，ひとつのトレース命令として扱われる．

6.2.13 U（トレースなし実行）コマンド

Uコマンドは T コマンドと同じであるが，最初の１命令の前にしかＣＰＵステータスを表示しない．形式は，

U
Un
US
USn

である．nは実行する命令数である．Uコマンドは任意のキーをたたくことにより，途中で終了させることができる．

6.2.14 V（レイアウト表示）コマンド

Vコマンドは，最後にＥコマンドかＲコマンドでロードされたファイルについての情報を表示する．形式は，

V

である．

最後にロードされたファイルがＥコマンドによるものであれば，そのファイルのセグメントごとのスタート／エンド・アドレスを表示する．Ｒコマンドによるものであれば，ファイルの読込まれた領域のスタート／エンド・アドレスを表示する．ＲもＥもまだ使われていないときには，？マークが出力される．

6.2.15 W（書込み）コマンド

Wコマンドは，メモリ上のある連続領域の内容を，ディスク・ファイルに書込む．形式は，

W<ファイル名>
W<ファイル名>,s,f

である．<ファイル名> はデータを受け取るディスク・ファイルのファイル名と拡張名であり，sとfは書き込む領域のスタート／エンドの20ビット・アドレスである．fにセグメントが指定されていないときは，sのセグメントと同じとみなす．

最初の形式を使うと，ＤＤＴ-86は最後にＲコマンドで読込んだファイルのsとfの値を使用する．Ｒコマンドでロードしたファイルがないときには' クエスション・マーク（？）を出力する．最初の形式は，ファイルのパッチを行なっただけでファイルの長さが変化しないときに，ファイルに書き戻すのに有効である．

２番目の形式のsとfは，20ビット・アドレスで定義されるが，sの下位４ビットは無視される．つまり，書き出す領域は常に物理セグメント（パラグラフ）の境界から始まる．

Wコマンドで指定したファイル名がすでに存在しているときには，ＤＤＴ-86はまずそのファイルをデリートしてから新しいファイルを書く．

6.2.16 X（ＣＰＵステータスのチェック）コマンド

Xコマンドは，テストしているプログラムのＣＰＵステータスをチェックしたり，変更したりする．形式は，

X
Xr
Xf

である．rは8086ＣＰＵレジスタの名前，fはＣＰＵフラグの略語である．最初の形式を用いると，ＣＰＵステータスが次のように表示される．

AX　BX　CX ……… ES　IP
－－－－－－－－－ ×××× ×××× ××××………×××× ××××
〈命令〉

のはじめの9個のハイフンは，9個のCPUフラグのステータスを示す．各ハイフンは，対応するフラグがセットされていない（0）ことを示し，ハイフンのかわりにフラグ名の略語が表示されていれば，そのフラグはセットされている（1）ことを示す．フラグ名の略語を，表6-2に示す．

〈命令〉は，CSとIPレジスタの示している次に実行される予定の命令を，逆アセンブルしたものである．

表6-2　フラグ名の略語

略語	フラグ名
O	オーバフロー・フラグ
D	ディレクション・フラグ
I	インタラプト・イネーブル・フラグ
T	トラップフラグ
S	符号フラグ
Z	ゼロ・フラグ
A	補助キャリ・フラグ
P	パリティ・フラグ
C	キャリ・フラグ

2番目の形式では，テストされているプログラムでのCPUステータスのレジスタを変更することができる．Xにつづくrは16ビットのCPUレジスタのうちの1つの名前である．DDT-86はレジスタ名とその現在の値を表示してこれに答える．CRのみが入力されると，レジスタの値は変化しないが，ある値が入力されると，レジスタは入力された値に変更される．どちらの場合も，そのあとに次のレジスタが表示される．このプロセスがピリオドか不適切な値が入力されるまで，あるいは最後のレジスタになるまで続く．

3番目の形式では，テスト中のプログラムでのCPUステータスのフラグの1つを変更できる．DDT-86はフラグ名とそのステータスを表示してこれに答える．ある値が入力されるとフラグはその値に変更され，CRのみが返されるとフラグは変化しない．xfコマンドでは，チェック又は修正できるフラグは1つである．フラグのセット／リセットには1／0の値を入力する．

6.3　ディフォルト・セグメント値

ＤＤＴ-86は，いくつかのコマンドのオプションで使われる現セグメント値を保持している．ＤＤＴ-86のコマンド群は，次の２つのグループにわけられる．セグメントの指定がコマンド標準のものと，セグメントを特に指定しなくてもよいものとである．

前者はコード・セグメントに関係するもので，Ａ，Ｌ，Ｗなどである．これらのコマンドは指定されていないときは，タイプ１のセグメント値を使う．

ＤＤＴ-86が起動されたとき，タイプ１のセグメント値は０にセットされる．そして以下の動作が行われたときに変更される．

○Ｅコマンドでファイルがロードされたとき，ＤＤＴ-86はタイプ１のセグメント値をＣＳレジスタの値にセットする．
○Ｒコマンドによってファイルがロードされたとき，ＤＤＴ-86は読込まれたファイルのベース・セグメントの値を，タイプ１のセグメント値にセットする．
○Ｘコマンドで，ＣＳレジスタの値を変更したとき，ＤＤＴ-86は新しいＣＳの値をタイプ１のセグメント値にセットする．
○Ｇ，Ｔ，Ｕコマンドが使われた後，ＤＤＴ-86に制御が戻ったとき，ＣＳレジスタの値がタイプ１のセグメント値にセットされる．
○ＡコマンドヌはＬコマンドでセグメント値が指定されたとき，タイプ１のセグメント値はその値にセットされる．

後者のコマンドはデータ・セグメントに関係するもので，Ｄ，Ｆ，Ｍ，Ｓなどである．これらのコマンドは，内部で保持されているタイプ２のセグメント値を用いる．

ＤＤＴ-86が起動されたとき，タイプ２のセグメント値は０にセットされる．それが変更されるのは次のような場合である．

○ファイルがＥコマンドでロードされたとき，タイプ２のセグメント値はＤＳレジスタの値にセットされる．
○ファイルがＲコマンドでロードされたとき，タイプ２のセグメント値は読込まれたファイルのベース・セグメントの値にセットされる．
○ＸコマンドでＤＳレジスタが変更されたとき，タイプ２のセグメント値は新しいＤＳの値になる．
○Ｇ，Ｔ，Ｕコマンドが実行された後，ＤＤＴ-86に制御が戻ったとき，タイプ２のセグメント値はＤＳレジスタの値となる．
○Ｄ，Ｆ，Ｍ，Ｓコマンドでセグメント値が指定されたとき，タイプ２のセグメント値はその値にセットされる．

ＣＳとＤＳレジスタの値を使うプログラムが評価されるとき，ＤＤＴ-86のすべてのコマンドは，セグメント越えの指定のない限り，同じセグメント値をデフォルトとする．

ＧコマンドはＣＳレジスタをディフォルトとするので，どちらのグループにも属さない．

表6-3　ＤＤＴ-86のディフォルト・セグメント値

コマンド	タイプ1	タイプ2
A	X	
D		X
E	U	U
F		X
G	U	U
H		
I		
L	X	

コマンド	タイプ1	タイプ2
M		X
R	U	U
S		X
T	U	U
U	U	U
V		
W	X	
X	U	U

Ｘ＝指定がなければ，このデフォルトを用い，指定があればそれに変る．
Ｕ＝このデフォルトを更新する．

6.4 A／Lコマンドにおけるアセンブラの文法

A／Lコマンドで通常使われるアセンブラ文の文法は，基本的には8086の標準アセンブラ言語と同じである．若干の例外があるので次に述べる．

○DDT-86では，入力された数値はすべて，16進数とみなす．

○3種類のプレフィクス（LOCK，繰返し命令，セグメント越え）は，文の中に現れてもよいが，文の命令コードの前になくてはならない．1種類のプレフィクスが1行に複数あってもよい．

○バイトとワードのストリング命令の区別は次のように行う．

バイト	ワード
LODSB	LODSW
STOSB	STOSW
SCASB	SCASW
MOVSB	MOVSW
CMPSB	CMPSW

○制御移動命令のニーモニックは，次のようになる．

ショート	ノーマル	ファー
JMPS	JMP	JMPF
	CALL	CALLF
	RET	RETF

○CALLFやJMPFのオペランドが20ビットの絶対アドレスのときは，次の形で入力する．

ssss：oooo

ssssはセグメント値，ooooはオフセット値である．

○バイトでもワードでもよいオペランドはまぎらわしいため，プレフィクス「BYTE」と「WORD」を付ける．これらのプレフィクスは「BY」「WO」と略せる．

```
INC     BYTE [BP]
NOT     WORD [1234]
```

必要なときにこれを忘れると，エラーメッセージが出力される．

○オペランドが直接アドレスのとき，数値と区別するために大かっこで囲む．

```
ADD     AX,5      ;add 5 to register AX
ADD     AX,[5]    ;add the contents of location 5 to AX
```

○レジスタ間接修飾オペランドは，

〔ポインタ・レジスタ〕
〔インデックス・レジスタ〕
〔ポインタ+インデックス・レジスタ〕

という形となる．ポインタ・レジスタはBXとBP，インデックス・レジスタはSIとDIである．これらの形のオペランドの前に数値のオフセットを置くことができる．

```
ADD     BX,[BP+SI]
ADD     BX,3[BP+SI]
ADD     BX,1D47[BP+SI]
```

6.5 DDT-86の実行例

次の実行例は，ユーザが簡単なソートのプログラムを会話的にデバックしているものである．イタリック体の注釈は，それに続く1連のステップの説明である．

テストするプログラムのソース・ファイル

```
A>type sort.a86
;
;       simple sort program
;
sort:
        mov     si,0            ;initialize index
        mov     bx,offset nlist ;bx = base of list.
        mov     sw,0            ;clear switch flag
comp:
        mov     al,[bx+si]      ;get byte from list
        cmp     al,1[bx+si]     ;compare with next byte
        jna     inci            ;don't switch if in order
        xchg    al,1[bx+si]     ;do first part of switch
        mov     [bx+si],al      ;do second part
        mov     sw,1            ;set switch flag
inci:
        inc     si              ;increment index
        cmp     si,count        ;end of list?
        jnz     comp            ;no, keep going
        test    sw,1            ;done - any switches?
        jnz     sort            ;yes, sort some more
done:
        jmp     done            ;get here when list ordered
;
        dseg
        org     100h            ;leave space for base page
;
nlist   db      3,8,4,6,31,6,4,1
count   equ     offset $ - offset nlist
sw      db      0
        end
```

プログラムのアセンブル

```
A>asm86 sort

CP/M 8086 ASSEMBLER VER 1.1
END OF PASS 1
END OF PASS 2
END OF ASSEMBLY. NUMBER OF ERRORS:   0
```

ASM-86で作られたリスティング・ファイルの出力

```
A>type sort.lst
CP/M ASM86 1.1  SOURCE: SORT.A86                                    PAGE    1


                        ;
                        ;       simple sort program
                        ;
                        sort:
 0000 BE0000                    mov     si,0            ;initialize index
 0003 BB0001                    mov     bx,offset nlist ;bx = base of list
 0006 C606080100                mov     sw,0            ;clear switch flag
                        comp:
 000B 8A00                      mov     al,[bx+si]      ;get byte from list
 000D 3A4001                    cmp     al,1[bx+si]     ;compare with next byte
 0010 760A                      jna     inci            ;don't switch if in order
 0012 864001                    xchg    al,1[bx+si]     ;do first part of switch
 0015 8800                      mov     [bx+si],al      ;do second part
 0017 C606080101                mov     sw,1            ;set switch flag
                        inci:
 001C 46                        inc     si              ;increment index
 001D 83FE08                    cmp     si,count        ;end of list?
 0020 75E9                      jnz     comp            ;no, keep going
 0022 F606080101                test    sw,1            ;done - any switches?
 0027 75D7                      jnz     sort            ;yes, sort some more
                        done:
 0029 E9FDFF                    jmp     done            ;get here when list ordered
                        ;
                                dseg
                                org     100h            ;leave space for base page
                        ;
```

```
0100 030804061F06 nlist     db      3,8,4,6,31,6,4,1
     0401
  0008                count     equ     offset $ - offset nlist
0108 00               sw        db      0
                                end


END OF ASSEMBLY. NUMBER OF ERRORS:   0
```

ASM-86で作られたシンボル・テーブルの出力

```
A>type sort.sym
0000 VARIABLES
0100 NLIST          0108 SW

0000 NUMBERS
0008 COUNT

0000 LABELS
000B COMP           0029 DONE         001C INCI         0000 SORT
```

ASM-86で作られた16進形式ファイルの出力

```
A>type sort.h86
:0400000300000000F9
:1B000081BE0000BB0001C6060801008A003A4001760A8640018800C60608016C
:11001B81014683FE0875E9F60608010175D7E9FDFFEE
:09010082030804061F0604010035
:00000001FF
```

.H86ファイルよりCMDファイルを生成する.

```
A>gencmd sort

BYTES READ      0039
RECORDS WRITTEN 04
```

DDT-86を起動しSORT.CMDをロードする.

```
A>ddt86 sort
DDT86 1.0
     START       END
CS 047D:0000 047D:002F
DS 0480:0000 0480:010F
```

レジスタの初期値の表示.

```
-x
            AX   BX   CX   DX   SP   BP   SI   DI   CS   DS   SS   ES   IP
--------- 0000 0000 0000 0000 119E 0000 0000 0000 047D 0480 0491 0480 0000
MOV     SI,0000
```

コード・セグメントの頭の逆アセンブル.

```
-l
047D:0000 MOV     SI,0000
047D:0003 MOV     BX,0100
047D:0006 MOV     BYTE [0108],00
047D:000B MOV     AL,[BX+SI]
047D:000D CMP     AL,01[BX+SI]
047D:0010 JBE     001C
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0008
047D:0020 JNZ     000B
```

データ・セグメントの頭の表示.

```
-d100,10f
0480:0100 03 08 04 06 1F 06 04 01 00 00 00 00 00 00 00 00 ................
```

```
        コード・セグメントの残りの逆アセンブル.
-l
047D:0022 TEST    BYTE [0108],01
047D:0027 JNZ     0000
047D:0029 JMP     0029
047D:002C ADD     [BX+SI],AL
047D:002E ADD     [BX+SI],AL
047D:0030 DAS
047D:0031 ADD     [BX+SI],AL
047D:0033 ??=     6C
047D:0034 POP     ES
047D:0035 ADD     [BX],CL
047D:0037 ADD     [BX+SI],AX
047D:0039 ??=     6F

        ブレークポイントを29Hに設定して, IP (＝0) からプログラムを実行する.
-g,29
*047D:0029      ブレークポイントに出会う.

        ソート後のリストを表示.
-d100,10f
0480:0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

        正しい結果でないので, ファイルを再ロードする.
-esort
     START     END
CS 047D:0000 047D:002F
DS 0480:0000 0480:010F

        3命令だけトレースする.
-t3
           AX   BX   CX   DX   SP   BP   SI   DI   IP
------Z-P- 0000 0100 0000 0000 119E 0000 0008 0000 0000 MOV    SI,0000
------Z-P- 0000 0100 0000 0000 119E 0000 0000 0000 0003 MOV    BX,0100
------Z-P- 0000 0100 0000 0000 119E 0000 0000 0000 0006 MOV    BYTE [0108],00
*047D:000B

        さらにトレースする.
-t3
           AX   BX   CX   DX   SP   BP   SI   DI   IP
-----Z-P-  0000 0100 0000 0000 119E 0000 0000 0000 000B MOV    AL,[BX+SI]
-----Z-P-  0003 0100 0000 0000 119E 0000 0000 0000 000D CMP    AL,01[BX+SI]
-----S-A-C 0003 0100 0000 0000 119E 0000 0000 0000 0010 JBE    001C
*047D:001C

        ソートする前のリストを表示.
-d100,10f
0480:0100 03 08 04 06 1F 06 04 01 00 00 00 00 00 00 00 00 ................

        次に実行される命令を表示.
-l
047D:001C INC     SI
047D:001D CMP     SI,0008
047D:0020 JNZ     000B
047D:0022 TEST    BYTE [0108],01
047D:0027 JNZ     0000
047D:0029 JMP     0029
047D:002C ADD     [BX+SI],AL
047D:002E ADD     [BX+SI],AL
047D:0030 DAS
047D:0031 ADD     [BX+SI],AL
047D:0033 ??=     6C
047D:0034 POP     ES

        さらに3命令をトレース.
-t3
           AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-A-C  0003 0100 0000 0000 119E 0000 0000 0000 001C INC    SI
---------C 0003 0100 0000 0000 119E 0000 0001 0000 001D CMP    SI,0008
----S-APC  0003 0100 0000 0000 119E 0000 0001 0000 0020 JNZ    000B
*047D:000B
```

現在の IPから命令を表示.

```
-l
047D:000B MOV     AL,[BX+SI]
047D:000D CMP     AL,01[BX+SI]
047D:0010 JBE     001C
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0008
047D:0020 JNZ     000B
047D:0022 TEST    BYTE [0108],01
047D:0027 JNZ     0000
047D:0029 JMP     0029

-t3
               AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-APC 0003 0100 0000 0000 119E 0000 0001 0000 000B MOV     AL,[BX+SI]
----S-APC 0008 0100 0000 0000 119E 0000 0001 0000 000D CMP     AL,01[BX+SI]
--------- 0008 0100 0000 0000 119E 0000 0001 0000 0010 JBE     001C
*047D:0012

-l
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0008
047D:0020 JNZ     000B
047D:0022 TEST    BYTE [0108],01
047D:0027 JNZ     0000
047D:0029 JMP     0029
047D:002C ADD     [BX+SI],AL
047D:002E ADD     [BX+SI],AL
047D:0030 DAS
```

入れ替えが行なわれるまで実行.

```
-g,20
*047D:0020
```

リストを表示.

```
-d100,10f
0480:0100 03 04 08 06 1F 06 04 01 01 00 00 00 00 00 00 00 ................
```

4と8はうまく入れ替っている

```
-t
               AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-APC 0004 0100 0000 0000 119E 0000 0002 0000 0020 JNZ     000B
*047D:000B
```

次の命令を表示.

```
-l
047D:000B MOV     AL,[BX+SI]
047D:000D CMP     AL,01[BX+SI]
047D:0010 JBE     001C
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0008
047D:0020 JNZ     000B
047D:0022 TEST    BYTE [0108],01
047D:0027 JNZ     0000
047D:0029 JMP     0029
```

入れ替えは正常に動いているので, 再ロードして境界条件をチェックしようとしている.

```
-esort
     START         END
CS 047D:0000  047D:002F
DS 0480:0000  0480:010F
```

速くするためにリストの長さを３にする．

```
-ald
047D:001D cmp si,3
047D:0020
```

ソート前のデータ・リストを表示．

```
-d100
0480:0100 03 08 04 06 1F 06 04 01 00 00 00 00 00 00 00 00 ................
0480:0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0480:0120 00 00 00 00 00 00 00 00 00 00 00 00 00 20 20 20 .............
```

はじめの３要素がソートされたところにブレークポイントを設定．

```
-g,29
*047D:0029
```

ソートされているかどうかを見る．

```
-d100,10f
0480:0100 03 04 06 08 1F 06 04 01 00 00 00 00 00 00 00 00 ................
```

なんと４番目の要素までソートされている．

```
-esort
     START     END
CS 047D:0000 047D:002F
DS 0480:0000 0480:010F
```

もういちどトレースしてみる．

```
-ald
047D:001D cmp si,3
047D:0020 .

-t9
          AX   BX   CX   DX   SP   BP   SI   DI   IP
-----Z-P- 0006 0100 0000 0000 119E 0000 0003 0000 0000 MOV   SI,0000
-----Z-P- 0006 0100 0000 0000 119E 0000 0000 0000 0003 MOV   BX,0100
-----Z-P- 0006 0100 0000 0000 119E 0000 0000 0000 0006 MOV   BYTE [0108],00
-----Z-P- 0006 0100 0000 0000 119E 0000 0000 0000 000B MOV   AL,[BX+SI]
-----Z-P- 0003 0100 0000 0000 119E 0000 0000 0000 000D CMP   AL,01[BX+SI]
----S-A-C 0003 0100 0000 0000 119E 0000 0000 0000 0010 JBE   001C
----S-A-C 0003 0100 0000 0000 119E 0000 0000 0000 001C INC   SI
--------C 0003 0100 0000 0000 119E 0000 0001 0000 001D CMP   SI,0003
----S-A-C 0003 0100 0000 0000 119E 0000 0001 0000 0020 JNZ   000B
*047D:000B

-l
047D:000B MOV    AL,[BX+SI]
047D:000D CMP    AL,01[BX+SI]
047D:0010 JBE    001C
047D:0012 XCHG   AL,01[BX+SI]
047D:0015 MOV    [BX+SI],AL
047D:0017 MOV    BYTE [0108],01
047D:001C INC    SI
047D:001D CMP    SI,0003
047D:0020 JNZ    000B
047D:0022 TEST   BYTE [0108],01
047D:0027 JNZ    0000
047D:0029 JMP    0029

-t3
          AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-A-C 0003 0100 0000 0000 119E 0000 0001 0000 000B MOV   AL,[BX+SI]
----S-A-C 0008 0100 0000 0000 119E 0000 0001 0000 000D CMP   AL,01[BX+SI]
--------- 0008 0100 0000 0000 119E 0000 0001 0000 0010 JBE   001C
*047D:0012

-l
047D:0012 XCHG   AL,01[BX+SI]
047D:0015 MOV    [BX+SI],AL
047D:0017 MOV    BYTE [0108],01
047D:001C INC    SI
047D:001D CMP    SI,0003
047D:0020 JNZ    000B
047D:0022 TEST   BYTE [0108],01
```

```
-t3
             AX   BX   CX   DX   SP   BP   SI   DI   IP
---------- 0008 0100 0000 0000 119E 0000 0001 0000 0012 XCHG  AL,01[BX+SI]
---------- 0004 0100 0000 0000 119E 0000 0001 0000 0015 MOV   [BX+SI],AL
---------- 0004 0100 0000 0000 119E 0000 0001 0000 0017 MOV   BYTE [0108],01
*047D:001C

-d100,10f
0480:0100 03 04 08 06 1F 06 04 01 01 00 00 00 00 00 00 00 ................
```

ここまではOK.

```
-t3
             AX   BX   CX   DX   SP   BP   SI   DI   IP
---------- 0004 0100 0000 0000 119E 0000 0001 0000 001C INC   SI
---------- 0004 0100 0000 0000 119E 0000 0002 0000 001D CMP   SI,0003
----S-APC  0004 0100 0000 0000 119E 0000 0002 0000 0020 JNZ   000B
*047D:000B

-l
047D:000B MOV    AL,[BX+SI]
047D:000D CMP    AL,01[BX+SI]
047D:0010 JBE    001C
047D:0012 XCHG   AL,01[BX+SI]
047D:0015 MOV    [BX+SI],AL
047D:0017 MOV    BYTE [0108],01
047D:001C INC    SI
047D:001D CMP    SI,0003
047D:0020 JNZ    000B
047D:0022 TEST   BYTE [0108],01
047D:0027 JNZ    0000
047D:0029 JMP    0029

-t3
             AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-APC  0004 0100 0000 0000 119E 0000 0002 0000 000B MOV   AL,[BX+SI]
----S-APC  0008 0100 0000 0000 119E 0000 0002 0000 000D CMP   AL,01[BX+SI]
---------- 0008 0100 0000 0000 119E 0000 0002 0000 0010 JBE   001C
*047D:0012
```

あきらかに，3番目と4番目の要素が比較されている．再ロードする．

```
-esort
    START          END
CS 047D:0000 047D:002F
DS 0480:0000 0480:010F

-l
047D:0000 MOV    SI,0000
047D:0003 MOV    BX,0100
047D:0006 MOV    BYTE [0108],00
047D:000B MOV    AL,[BX+SI]
047D:000D CMP    AL,01[BX+SI]
047D:0010 JBE    001C
047D:0012 XCHG   AL,01[BX+SI]
047D:0015 MOV    [BX+SI],AL
047D:0017 MOV    BYTE [0108],01
047D:001C INC    SI
047D:001D CMP    SI,0008
047D:0020 JNZ    000B
```

長さを修正．

```
-ald
047D:001D cmp si,7
047D:0020 .
```

再実行．

```
-g,29
*047D:0029
```

リストがソートされているかどうかを見る.

```
-d100,10f
0480:0100 01 03 04 04 06 06 08 1F 00 00 00 00 00 00 00 00 ................
```

OKなのでディスク・ファイルを正しく直す. そのためにはCMDファイルをヘッダつきでロードする必要がある. そこでRコマンドを用いる.

```
-rsort.cmd
 START       END
2000:0000 2000:01FF
```

はじめの80Hバイトにヘッダが入っているので, コード・セグメントは80Hから始まっている.

```
-l80
2000:0080 MOV     SI,0000
2000:0083 MOV     BX,0100
2000:0086 MOV     BYTE [0108],00
2000:008B MOV     AL,[BX+SI]
2000:008D CMP     AL,01[BX+SI]
2000:0090 JBE     009C
2000:0092 XCHG    AL,01[BX+SI]
2000:0095 MOV     [BX+SI],AL
2000:0097 MOV     BYTE [0108],01
2000:009C INC     SI
2000:009D CMP     SI,0008
2000:00A0 JNZ     008B
```

修正する.

```
-a9d
2000:009D cmp si,7
```

ディスク・ファイルに書き戻す(ファイルの長さは変わらない).

```
-wsort.cmd
```

ファイルを再ロード.

```
-esort

      START       END
CS 047D:0000 047D:002F
DS 0480:0000 0480:010F
```

修正が効いているか検証する.

```
-l
047D:0000 MOV     SI,0000
047D:0003 MOV     BX,0100
047D:0006 MOV     BYTE [0108],00
047D:000B MOV     AL,[BX+SI]
047D:000D CMP     AL,01[BX+SI]
047D:0010 JBE     001C
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0007
047D:0020 JNZ     000B
```

実行.

```
-g,29
*047D:0029
```

OKなので終了する.

```
-d100,10f
0480:0100 01 03 04 04 06 06 08 1F 00 00 00 00 00 00 00 00 ................
-^C
A>
```

付録A
ＡＳＭ-86の起動

コマンド：　ＡＳＭ86

構文：　ＡＳＭ86 <ファイル名> [$ <パラメータ>]

<ファイル名> は8086のアセンブラ・ソース・ファイルで，ドライブ名と拡張名はオプションである．拡張名のディフォルトは，.A86．

<パラメータ> は１文字のタイプと１文字のデバイス名から成るパラメータ列である．これらは下表に示す．

パラメータ：　$ Td

Ｔはタイプ，ｄはデバイス名である．

表Ａ-１　パラメータのタイプとデバイス

デバイス	パラメータ				
	A	H	P	S	F
A～P	X	X	X	X	
X		X	X	X	
Y		X	X	X	
Z		X	X	X	
I					X
D					d

Ｘ＝指定可能
ｄ＝ディフォルト

Ｆタイプを除いて，ディフォルト・デバイスはカレント・ディフォルト・ドライブである．

表Ａ-２　パラメータ・タイプ

A	アセンブラ・ソース・ファイルの制御
H	16進形式ファイルの制御
P	プリント・ファイルの制御
S	シンボル・ファイルの制御
F	16進ファイルの出力フォーマット

表A-3　デバイス指定

A～P	ドライブAからPまで
X	コンソール
Y	プリンタ
Z	出力なし
I	Intel 16進形式
D	Digital Research 16進形式

表A-4　呼び出し例

ASM86 IO	IO.A86をアセンブルし，IO.HEX，IO.LST，IO.SYMを生成する．
ASM86 IO.ASM $ AD SZ	ドライブDにあるIO.ASMをアセンブルし，IO.LST とIO.HEXを生成する．シンボル・ファイルは生成しない．
ASM86 IO $ PY SX	IO.A86をアセンブルし，IO.HEXを生成し，リストを直接プリンタに，シンボル・テーブルをコンソールに出力する．
ASM86 IO $ FD	Digital Researchの16進形式を生成する．
ASM86 IO $ FI	Intelの16進形式を生成する．

付録B
Intel アセンブラとの相違点

CP／Mの8086アセンブラは，ジャンプ，コール，リターン命令の遠近を明示的に指定する点を除いて，Intel8086アセンブラと同じニーモニックを使っている．次にその違いを示す．

表B-1　ニーモニックの相違

ニーモニックの機能	CP／M	INTEL
セグメント内のショート・ジャンプ	JMPS	JMP
セグメント外へのジャンプ	JMPF	JMP
セグメント外へのリターン	RETF	RET
セグメント外のコール	CALLF	CALL

付録C
ＡＳＭ-86の16進出力形式

ユーザの選択によって，ＡＳＭ-86はマシン語コードを，IntelとDigital Researchのどちらの16進形式ででも生成できる．Intel形式は8086用にIntelが定義した形式と同じである．Digital Research形式はIntel形式とほとんど同じであるが，セグメント情報の16進レコードが追加されている．どちらの形式の出力もＧＥＮＣＭＤに対して入力できるが，Digital Research形式は自動的にセグメント情報を準備する．セグメントとは，再配置可能なプログラムの最小単位である．

表Ｃ-１は16進レコードの各バイトのレイアウトである．各レコードは，表Ｃ-２に示す４つの形式のうちの１つである．次に16進レコードの例を挙げる．

```
バイトNO.  0 1 2 3 4 5 6 7 8 9 ..............n
内容       : l l a a a a t t d d d .........c c CR LF
```

表Ｃ-１　16進レコードの内容

バ イ ト	内　容	記　号
0	レコード・マーク	:
1～2	レコード長	ll
3～6	ロード・アドレス	aaaa
7～8	レコード・タイプ	tt
9～(n－1)	データ・タイプ	dddd…d
n～(n＋1)	チェック・サム	cc
n＋2	キャリッジ・リターン	ＣＲ
n＋3	ライン・フィード	ＬＦ

表Ｃ-２　16進レコード形式

レコード・タイプ	内　容	フォーマット
00	データ・レコード	:ll aaaa ＤＴ <データ‥> cc
01	エンド・オブ・ファイル	:00 0000 01 ＦＦ
02	拡張アドレス・マーク	:02 0000 ST ssss cc
03	スタート・アドレス	:04 0000 03 ssss iiii cc

ll＝レコード長．データのバイト数．
cc＝チェック・サム．全レコード・バイトの合計．
aaaa＝16ビット・アドレス．
ssss＝16ビット・セグメント．
iiii＝スタート・アドレスのオフセット．
ＤＴ＝データ・レコード・タイプ．
ＳＴ＝セグメント・アドレス・レコード・タイプ．

レコードタイプ00と02の定義では，IntelとDigital Researchの16進形式に差がある．Intelでは，データ・レコード・タイプとセグメント・アドレス・タイプの値はひとつである．Digital Researchでは，各レコードは，表C-3に示すように，それを含むセグメントに依る．

表C-3　セグメント・レコード・タイプ

記号	Intel	Digital	意　味
DT	00		データは8086の全セグメントに属する．
		81H	コード・セグメントに属する．
		82H	データ・セグメントに属する．
		83H	スタック・セグメントに属する．
		84H	エクストラ・セグメントに属する．
ST	02		全セグメント・アドレス・レコード．
		85H	コード・セグメント絶対アドレス．
		86H	データ・セグメントアドレス．
		87H	スタック・セグメント・アドレス．
		88H	エクストラ・セグメント・アドレス

付録D

予約語

表D-1　予約語

定義済み数値				
BYTE	WORD	DWORD		
		Operators		
EQ	GE	GT	LE	LT
NE	OR	AND	MOD	NOT
PTR	SEG	SHL	SHR	XOR
LAST	TYPE	LENGTH	OFFSET	
		アセンブラ擬似命令		
DB	DD	DW	IF	RS
RB	RW	END	ENDM	EQU
ORG	CSEG	DSEG	ESEG	SSEG
EJECT	ENDIF	TITLE	LIST	NOLIST
INCLUDE	SIMFORM	PAGESIZE	CODEMACRO	PAGEWIDTH
		コード・マクロ擬似命令		
DB	DD	DW	DBIT	RELB
RELW	MODRM	SEGFIX	NOSEGFIX	
		8086レジスタ		
AH	AL	AX	BH	BL
BP	BX	CH	CL	CS
CX	DH	DI	DL	DS
DX	ES	SI	SP	SS
		命令ニーモニック－付録E参照		

付録E
ＡＳＭ-86命令の要約

表E-1　ＡＳＭ-86命令の要約

ニーモニック	説　　明	節
AAA	加算用にASCIIに調整	4.3
AAD	除算用にASCIIに調整	4.3
AAM	乗算用にASCIIに調整	4.3
AAS	減算用にASCIIに調整	4.3
ADC	キャリ付き加算	4.3
ADD	加算	4.3
AND	論理積	4.3
CALL	セグメント内コール	4.5
CALLF	セグメント外コール	4.5
CBW	バイト→ワード変換	4.3
CLC	キャリのクリア	4.6
CLD	ディレクション・フラグのクリア	4.6
CLI	割込みフラグのクリア	4.6
CMC	キャリの反転	4.6
CMP	比較	4.3
CMPS	ストリングの比較（バイト又はワード）	4.4
CWD	ワード→２ワードの変換	4.3
DAA	加算用に10進に整合	4.3
DAS	減算用に10進に整合	4.3
DEC	減少	4.3
DIV	除算	4.6
ESC	エスケープ	4.6
HLT	１時停止	4.3
IDIV	整数除算	4.3
IMUL	整数乗算	4.2
IN	入力(バイト又はワード)	4.3
INC	増加	4.5
INT	割込み	4.5
INTO	オーバフロー割込み	4.5
IRET	割込みリターン	4.5
JA	上のときジャンプ	4.5
JAE	上か等しいときジャンプ	4.5
JB	下のときジャンプ	4.5
JBE	下か等しいときジャンプ	4.5
JC	キャリのときジャンプ	4.5
JCXZ	ＣＸがゼロのときジャンプ	4.5
JE	等しいときジャンプ	4.5
JG	大きいときジャンプ	4.5
JGE	大きいか等しいときジャンプ	4.5

表E-1 (続き)

ニーモニック	説　　明	節
JL	小さいときジャンプ	4.5
JLE	小さいか等しいときジャンプ	4.5
JMP	ジャンプ (セグメント内)	4.5
JMPF	ジャンプ (セグメント外)	4.5
JMPS	ジャンプ (変位8ビット)	4.5
JNA	上でないときジャンプ	4.5
JNAE	上でないか等しいときジャンプ	4.5
JNB	下でないときジャンプ	4.5
JNBE	下でないか等しいときジャンプ	4.5
JNC	キャリでないときジャンプ	4.5
JNE	等しくないときジャンプ	4.5
JNG	大きくないときジャンプ	4.5
JNGE	大きくないか等しいときジャンプ	4.5
JNL	小さくないときジャンプ	4.5
JNLE	小さくないか等しいときジャンプ	4.5
JNO	オーバフローでないときジャンプ	4.5
JNP	パリティでないときジャンプ	4.5
JNS	符号なしのときジャンプ	4.5
JNZ	ゼロでないときジャンプ	4.5
JO	オーバフローのときジャンプ	4.5
JP	パリティのときジャンプ	4.5
JPE	偶数パリティのときジャンプ	4.5
JPO	奇数パリティのときジャンプ	4.5
JS	符号つきのときジャンプ	4.5
JZ	ゼロのときジャンプ	4.5
LAHF	フラグをAHへロード	4.2
LDS	ポインタをDSへロード	4.2
LEA	実効アドレスをロード	4.2
LES	ポインタをESへロード	4.2
LOCK	バスをロック	4.6
LODS	ストリングのロード(バイト又はワード)	4.4
LOOP	ループ	4.5
LOOPE	等しい間ループ	4.5
LOOPNE	等しくない間ループ	4.5
LOOPNZ	ゼロでない間ループ	4.5
LOOPZ	ゼロの間ループ	4.5
MOV	転送	4.2
MOVS	ストリングの転送(バイト又はワード)	4.4
MUL	乗算	4.3
NEG	負値化	4.3
NOT	否定	4.3
OR	論理和	4.3
OUT	出力(バイト又はワード)	4.2
POP	スタックからの取り出し	4.2

表E-1（続き）

ニーモニック	説　　明	節
POPF	フラグをスタックより取り出し	4.2
PUSH	スタックへプッシュ	4.2
PUSHF	フラグをスタックへプッシュ	4.2
RCL	キャリを介して左へ回転	4.3
RCR	キャリを介して右へ回転	4.3
REP	繰返し	4.4
RET	リターン（セグメント内）	4.5
RETF	リターン（セグメント外）	4.5
ROL	左回転	4.3
ROR	右回転	4.3
SAHF	AHをフラグへ格納	4.2
SAL	算術的左シフト	4.3
SAR	算術的右シフト	4.3
SBB	けた下り減算	4.3
SCAS	ストリングの走査（バイト又はワード）	4.4
SHL	左シフト	4.3
SHR	右シフト	4.3
STC	キャリのセット	4.6
STD	ディレクション・フラグのセット	4.6
STI	割込みフラグのセット	4.6
STOS	ストリングのストア（バイト又はワード）	4.4
SUB	減算	4.3
TEST	テスト	4.3
WAIT	ウエイト	4.6
XCHG	入れ替え	4.2
XLAT	翻訳	4.2
XOR	排他的論理和	4.3

付録F
サンプル・プログラム

```
CP/M ASM86 1.1  SOURCE: APPF.A86      Terminal Input/Output          PAGE
 1


                        title 'Terminal Input/Output'
                        pagesize 50
                        pagewidth 79
                        simform
                        ;
                        ;****** Terminal I/O subroutines ********
                        ;
                        ;       The following subroutines
                        ;       are included:
                        ;
                        ;       CONSTAT   -  console status
                        ;       CONIN     -  console input
                        ;       CONOUT    -  console output
                        ;
                        ;       Each routine requires CONSOLE NUMBER
                        ;       in the BL - register
                        ;
                        ;
                        ;       *****************
                        ;       *  Jump table:  *
                        ;       *****************
                        ;
                        CSEG            ; start of code segment
                        ;
                        jmp_tab:
0000 E90600                     jmp     constat
0003 E91900                     jmp     conin
0006 E92B00                     jmp     conout
                        ;
                        ;
                        ;       ***********************
                        ;       *  I/O port numbers   *
                        ;       ***********************
                        ;
```

```
                    ;
                    ;          Terminal 1:
                    ;
 0010               instat1          equ     10h     ; input status port
 0011               indata1          equ     11h     ; input port
 0011               outdata1         equ     11h     ; output port
 0001               readyinmask1     equ     01h     ; input ready mask
 0002               readyoutmask1    equ     02h     ; output ready mask
                    ;
                    ;          Terminal 2:
                    ;
 0012               instat2          equ     12h     ; input status port
 0013               indata2          equ     13h     ; input port
 0013               outdata2         equ     13h     ; output port
 0004               readyinmask2     equ     04h     ; input ready mask
 0008               readyoutmask2    equ     08h     ; output ready mask
                    ;
                    ;
                    ;        ***********
                    ;        * CONSTAT *
                    ;        ***********
                    ;
                    ;        Entry: BL - reg = terminal no
                    ;        Exit:  AL - reg = 0 if not ready
                    ;                          0ffh if ready
                    ;
                    constat:
0009 53E83F00                push bx ! call okterminal
                    constat1:
000D 52                      push dx
000E B600                    mov  dh,0                    ; read status port
0010 8A17                    mov  dl,instatustab [BX]
0012 EC                      in   al,dx
0013 224706                  and  al,readyinmasktab [bx]
0016 7402                    jz   constatout
0018 B0FF                    mov  al,0ffh
```

```
                          constatout:
001A 5A5B0AC0C3                     pop dx ! pop bx ! or al,al ! ret
                          ;
                          ;
                          ;         *********
                          ;         * CONIN *
                          ;         *********
                          ;
                          ;         Entry: BL - reg = terminal no
                          ;         Exit:  AL - reg = read character
                          ;
001F 53E82900             conin:    push bx ! call okterminal !
0023 E8E7FF               coninl:   call constatl                    ; test status
0026 74FB                           jz   coninl
0028 52                             push dx                          ; read character
0029 B600                           mov  dh,0
002B 8A5702                         mov  dl,indatatab [BX]
002E EC                             in   al,dx
002F 247F                           and  al,7fh                      ; strip parity bit
0031 5A5BC3                         pop dx ! pop bx ! ret
                          ;
                          ;
                          ;         **********
                          ;         * CONOUT *
                          ;         **********
                          ;
                          ;
                          ;         Entry: BL - reg = terminal no
                          ;                AL - reg = character to print
                          ;
0034 53E81400             conout:   push bx ! call okterminal
0038 52                             push dx
0039 50                             push ax
003A B600                           mov  dh,0                        ; test status
003C 8A17                           mov  dl,instatustab [BX]
                          conoutl:
003E EC                             in   al,dx
```

```
 003F 224708               and  al,readvoutmasktab [BX]
 0042 74FA                 jz   conoutl
 0044 58                   pop  ax                    ; write byte
 0045 8A5704               mov  dl,outdatatab [BX]
 0048 EE                   out  dx,al
 0049 5A5BC3               pop dx ! pop bx ! ret
                       ;
                       ;
                       ;   ++++++++++++++
                       ;   + OKTERMINAL +
                       ;   ++++++++++++++
                       ;
                       ;   Entry:  BL - reg = terminal no
                       ;
                       okterminal:
 004C 0ADB                 or   bl,bl
 004E 740A                 jz   error
 0050 80FB03               cmp  bl,length instatustab + 1
 0053 7305                 jae  error
 0055 FECB                 dec  bl
 0057 B700                 mov  bh,0
 0059 C3                   ret
                       ;
 005A 5B5BC3           error: pop bx ! pop bx ! ret     ; do nothing
                       ;
                       ;************** end of code segment ***************
                       ;
                       ;   ****************
                       ;   * Data segment *
                       ;   ****************
                       ;
                           dseg
                       ;   **************************
                       ;   * Data for each terminal *
                       ;   **************************
```

```
                ;

0000 1012       instatustab     db      instat1,instat2
0002 1113       indatatab       db      indata1,indata2
0004 1113       outdatatab      db      outdata1,outdata2
0006 0104       readyinmasktab  db      readyinmask1,readyinmask2
0008 0208       readyoutmasktab db      readyoutmask1,readyoutmask2
                ;
                ;*************** end of file ***********************
                end

END OF ASSEMBLY. NUMBER OF ERRORS:  0
```

付録G
コード・マクロ定義の文法

```
<codemacro> ::= CODEMACRO <name> [<formal$list>]
                [<list$of$macro$directives>]
                ENDM

<name> ::= IDENTIFIER

<formal$list> ::= <parameter$descr>[{,<parameter$descr>}]

<parameter$descr> ::= <form$name>:<specifier$letter>
                      <modifier$letter>[(<range>)]

<specifier$letter> ::= A | C | D | E | M | R | S | X

<modifier$letter> ::= b | w | d | sb

<range> ::= <single$range>|<double$range>

<single$range> ::= REGISTER | NUMBERB

<double$range> ::= NUMBERB,NUMBERB | NUMBERB,REGISTER |
                   REGISTER,NUMBERB | REGISTER,REGISTER

<list$of$macro$directives> ::= <macro$directive>
                               {<macro$directive>}

<macro$directive> ::= <db> | <dw> | <dd> | <segfix> |
                      <nosegfix> | <modrm> | <relb> |
                      <relw> | <dbit>

<db> ::= DB NUMBERB | DB <form$name>

<dw> ::= DW NUMBERW | DW <form$name>

<dd> ::= DD <form$name>

<segfix> ::= SEGFIX <form$name>

<nosegfix> ::= NOSEGFIX <form$name>

<modrm> ::= MODRM NUMBER7,<form$name> |
            MODRM <form$name>,<form$name>

<relb> ::= RELB <form$name>

<relw> ::= RELW <form$name>

<dbit> ::= DBIT <field$descr>{,<field$descr>}
```

```
<field$descr> ::= NUMBER15 ( NUMBERB ) |
                  NUMBER15 ( <form$name> ( NUMBERB ) )

<form$name> ::= IDENTIFIER
```

NUMBERB is 8-bits
NUMBERW is 16-bits
NUMBER7 are the values 0, 1,. . , 7
NUMBER15 are the values 0, 1,. . , 15

付録H
ＡＳＭ-86エラー・メッセージ

ＡＳＭ-86の出すエラー・メッセージには，致命的エラーと警告の２つの種類がある．致命的なエラーとは，ＡＳＭ-86がこれ以上アセンブルを続行できないエラーであり，警告はアセンブルされたプログラムの構文や意味についての報告である．次のメッセージは，ＡＳＭ-86がアセンブル中に，致命的エラーに出会ったことを示すものである．

```
NO FILE
DISK FULL
DIRECTORY FULL
DISK READ ERROR
CANNOT CLOSE
SYMBOL TABLE OVERFLOW
PARAMETER ERROR
```

ＡＳＭ-86は，構文や意味のエラーを，ASCII文字の数字で報告する．メッセージは，エラーのある行の直前に書かれる．１行に複数のエラーがある場合，最初の１つだけが報告される．表Ｈ-１にＡＳＭ-86の警告エラー・メッセージをまとめた．

表Ｈ-１　ＡＳＭ-86警告エラー・メッセージ

番号	意味
0	最初の項目が適当でない．
1	擬似命令がない．
2	不正な擬似命令．
3	変数を重複して定義した．
4	ラベルを重複して定義した．
5	未定義の命令．
6	行の終りにゴミがある——無視する．
7	命令とオペランドが合わない．
8	不正なオペランド．
9	命令がない．
10	式の要素が未定義．
11	擬似命令のオペランドが適当でない．
12	「IF」の入れ子は許されない——IFは無視する．
13	IFのオペランドがおかしい——IFは無視する．
14	IFとENDIFの数が合わない．
15	シンボルの不当な前方参照——そのままにしておく．
16	シンボルが重複して定義された——未定義として扱う．
17	命令がコード・セグメントにない．
18	ファイル名文法エラー．
19	INCLUDEの入れ子は許されない．
20	式の要素が不正．
21	オペランドの型情報がない．
22	ラベルが幅を越えた．
23	オペランドにセグメント情報がない．
24	コード・マクロ定義エラー．

付録I
ＤＤＴ-86エラー・メッセージ

表I-1　ＤＤＴ-86エラー・メッセージ

メッセージ	意　味
AMBIGUOUS OPERAND	まぎらわしいオペランドをもつ命令をアセンブルしようとした．オペランドの前にプレフィクス「ＢＹＴＥ」か「ＷＯＲＤ」を付けよ．
CANNOT CLOSE	Ｗコマンドで書き込んだファイルがクローズできない．
DISK READ ERROR	Ｒコマンドで指定されたファイルが正しく読めない．
DISK WRITE ERROR	Ｗコマンドによるディスクへの書き込みが正しく行われない．ディスクがいっぱいになった可能性もある．
INSUFFICIENT MEMORY	Ｒ又はＥコマンドで指定したファイルをロードするのにメモリが足りない．
MEMORY REQUEST DENIED	Ｒコマンドによるメモリへの要求が果たせない．メモリに一度に割り当てられるのは８ブロックまでである．
NO FILE	Ｒ又はＥコマンドで指定したファイルが，ディスクにない．
NO SPACE	Ｗコマンドで書込もうとしているファイルのためのディレクトリのスペースがない．
VERIFY ERROR AT S：O	Ｆ，Ｓ，Ｍ，Ａコマンドによってメモリに置かれた値が，正しく読み返せない．ＲＡＭ不良もしくはＲＯＭへ書き込もうとした，もしくは該当位置にメモリが存在しない．

ASM-86への追記

1）ＥＱＵ擬似命令における前方参照はエラーとなる．

2）コメント内の「！」は無視される．コメントはその行の物理的な終りまでである．

3）擬似命令の追加――ＩＦＬＩＳＴ及びＮＯＩＦＬＩＳＴ.「偽」となったＩＦブロックのリスティングを制御する．

4）ＩＦ擬似命令は５レベルまでのネストが可能になった．

5）新たにインプリメントされたニモニック

- ＪＣ，ＪＮＣ
- ＣＭＰＳＢ，ＣＭＰＳＷ，ＬＯＤＳＢ，ＬＯＤＳＷ，ＭＯＶＳＢ，ＭＯＶＳＷ，ＳＣＡＳＢ，ＳＣＡＳＷ，ＳＴＯＳＢ，ＳＴＯＳＷ

6）ＪＮＢＥ命令の障害は修正されている．

7）セグメント越えプリフィクスは，ストリング命令のソースオペランドの前にも許される．

8）式の中の比較演算子は，「真」の時は０ＦＦＦＦＨを返す．

9）不適当なコマンド・テイルが与えられた場合はアボートされる．

10）シンボル・テーブルのオーバーフローの際はアボートされる．

11）ディスク又はディレクトリ・フルの場合はアボートされる．

12）不完全なストリング（コーテーション不完全）はエラーとなる．

13）不適当な数値がＥＱＵ擬似命令中に現れた場合，エラーが報告される．

14）MP/M-86でのマルティプル・アクセスとの互換性のため，ソースファイルはＲ/Ｏモードでオープンされる．

15）.ＬＳＴファイルのフォーマットは次の通り．
- ファイルの先頭にフォーム・フィードが挿入される．
- ファイル末にはフォーム・フィードは挿入されない．
- 各ページの先頭にはＣＲ，ＬＦは挿入されない．
- 一頁当りの行数が変更された．
- コメントのスペースを確保するため，16進バイト間のスペースは削除される．
- ＮＯＬＩＳＴが指定されていても，エラーは出力される．
- リラティブ・インストラクションに対しても，アドレスフィールドには絶対番地で与えられる．

16）.ＳＹＭファイルのフォーマットは次の通り．
- ファイルの先頭にフォーム・フィードが挿入される．
- シンボルはグループ内でアルファベット順に並べられる．
- プリンタに送られる場合（＄ＳＹ），タブはスペースに展開される．

17）ＩＮＣＬＵＤＥ擬似命令で読みこまれるファイル．
- ファイルタイプのデフォルトは.A86.
- ファイルタイプ（拡張名）は３文字まで．
- インクルード・ファイルが発見されない時はアボートされる．
- ＄Ａスイッチが指定された場合，ソースファイルと同じドライブにあるものとみなされる．

18）ＩＮＣＬＵＤＥ擬似命令を用いたプログラムは，ＣＰ/Ｍ 1.4の下でも正常にアセンブルされる．

19）シンボルファイルに，約5.5Ｋ更に使えるようになった．

20）使用されたファクターが，アセンブルの終りに表示される（%欄．シンボルテーブルの使用状況を示す）．

21）処理速度が改善された（特に＄ＰＺが指定された時）．

DDT-86への追加

1）ユーザ・プログラムのスタックは，DDT-86のローカル・スタックではなく，デフォルトでCCPのスタックに設定される．

2）「;」で始まるコマンドラインは，コメントとして扱われる．

3）1ステップのトレースが行なわれている間は，割り込みが禁止される．

4）MP／M-86との互換性のため，BDOSエラーが発生した時はそのままBDOSにわたされる．

5）MP／M-86との互換性のため，リード及びロード後ファイルはクローズされる．

6）Bコマンド（Block Compare）が新たにインプリメントされた．パラメータはMコマンドと同様である．

FMシリーズ　CP/M-86編ユーザーズマニュアル

80SM-000020-1

発行日　1982年12月　初版

発行責任　©　富士通株式会社

・本書は，改善のため事前連絡なしに変更することがあります．

・なお，本書に記載されたデータの使用に起因する第3者の特許権その他の権利については，当社はその責を負いません．

・無断転載を禁じます．

・落丁，乱丁本はお取替えいたします．

Ⓣ

富士通